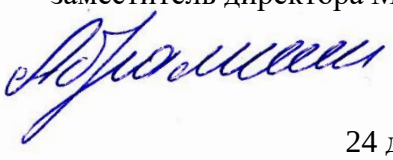
«Утверждаю»
Руководитель проекта,
заместитель директора МИЭМ НИУ ВШЭ


А.Е. Абрамешин

24 декабря 2021 года

Приложение № 8.1.

Учебное пособие по специальному курсу «Информационная безопасность»

Учебное пособие по специальному курсу

«Информационная безопасность»

Москва, НИУ ВШЭ
2021 г.

Оглавление

Введение

Учебное пособие по специальному курсу «Информационная безопасность» предназначено для получения обучающимися дополнительного образования в рамках курса по основам информационной безопасности.

Курс включает новые формы и методы обучения, его индивидуализацию с акцентом на самостоятельную работу и работу в команде. Актуальность программы связанна с широким распространением цифровых технологий, вследствие чего угрозы киберпространства могут коснуться каждого.

В результате изучения пособия учащиеся получат теоретические знания в области цифровизации, хранения и обработки информации, познакомятся с традиционными методами кодирования данных. На практике будут разобраны классические методы кодирования, системы хранения данных и устройство механизмов передачи данных.

Данное методическое пособие состоит из семи разделов. Отдельно представлены теоретические и практические материалы. К некоторым заданиям даны ответы в конце книги.

Раздел 1. Понятие информации, ее свойств. Состояние информационной безопасности

1.1. Что такое информация?

Начать стоит с понимания предмета – что такое информация, какой она бывает, что с ней делают. Данный разговор удобно проводить не в форме повествования, а в форме диалога: некоторые понятия известны ученикам из школьной программы, некоторые – из жизненного опыта.

Для работы с информацией необходимо сначала придать ей некоторую форму представления (то есть превратить её в данные), чтобы её можно было хранить, оценивать, обмениваться и совершать прочие процессы. Информация, сама по себе – в первую очередь интерпретация (смысл, идея) такого представления.

Первоначально слово «информация» трактуется как сведения, передаваемые людьми неким способом (устно или письменно, с помощью условных сигналов или технических средств и т. д.). С середины XX века на этот термин обратили внимание научные круги и стали его формализовать, так появилось общенаучное понятие «информация». Однако каждая отрасль все же по-своему его трактует, исходя из своей специфики. Идея передачи информации заключена в обмене сведениями или сигналами между некоторыми сторонами. Это могут быть люди, или человеком и техническое средство, или два технических средства сами по себе; такой обмен сигналами может быть и в животном и растительном мире. Например, говоря о генетической информации, биологи рассматривают передачу признаков от клетки к клетке, от организма к организму.

Сведения – это знания, передаваемые в виде сообщений, уведомлений и сигналов.

Данные – зарегистрированная информация; представление фактов, понятий или инструкций в форме, приемлемой для общения, интерпретации, или обработки человеком или с помощью автоматических средств (ISO/IEC/IEEE 24765-2010).

Информацию можно разделить на виды по различным критериям:
По способу восприятия:

- Визуальная – воспринимаемая органами зрения.
- Звуковая – воспринимаемая органами слуха.
- Тактильная – воспринимаемая тактильными рецепторами.
- Обонятельная – воспринимаемая обонятельными рецепторами.
- Вкусовая – воспринимаемая вкусовыми рецепторами.

По форме представления:

- Текстовая – передаваемая в виде символов, предназначенных обозначать лексемы языка.
- Числовая – в виде цифр и знаков, обозначающих математические действия.
- Графическая – в виде изображений, предметов, графиков.
- Звуковая – устная или в виде записи и передачи лексем языка аудиальным путём.
- Видеоинформация – в виде видеозаписи.

По назначению (это наиболее важный аспект в рамках рассматриваемого вопроса безопасности):

- Массовая – содержит тривиальные сведения и оперирует набором понятий, понятным большей части общества.
- Специальная – содержит специфический набор понятий, при использовании происходит передача сведений, которые могут быть не понятны каждому человеку, но необходимы и понятны в рамках узкой социальной группы, где используется данная информация.
- Секретная – передаваемая специально назначенному кругу лиц и по закрытым (то есть защищённым) каналам связи.
- Личная (приватная) – набор сведений о какой-либо личности, определяющий социальное

положение.

По значению (этот вид классификации стоит отдельно проговорить с примерами для четкого понимания):

- Актуальная – имеет практическую значимость лишь в данный момент времени.
- Достоверная – получена без искажений, в целости, из надежных проверенных источников, не искажает действительное положение дел.
- Понятная – выражена на языке, понятном тому, для кого она предназначена, в формате и виде, пригодном для восприятия.
- Полная – достаточная для понимания и анализа вопроса без додумываний и интерпретаций.
- Полезная – этот аспект субъективен и определяется лицом, для которого информация предназначена, в зависимости от объёма и возможностей её использования.
- Истинная.

1.2. Информационные процессы

Информационный поток увеличивается на 30% каждый год, вовлекая в свои процессы даже маленьких детей, которые чуть ли не с рождения пользуются телефонами и планшетами.

Информация со временем накапливается. Человеческое сознание уже не способно обрабатывать такие объемы. Тут на помощь приходят компьютеры, которые берут на себя функции по обработке и хранению сведений.

Информационный процесс – это совокупность последовательных действий, производимых над информацией для достижения определенного результата. К таким процессам относят процесс получения, создания, сбора, обработки, накопления, хранения, поиска, распространения, использования информации.

Далее рассмотрим все указанные процессы отдельно (согласно Федеральному Закону "Об информации, информатизации и защите информации" от 14 июня 2006 г.).

Получение информации – это процесс извлечения фактов и сведений о свойствах или взаимодействии объектов и явлений.

Создание информации – это процесс формирования кардинально новых сведений, фактов, не дефектных по форме и содержанию.

Сбор информации – это нахождение первичной информации для достижения определенной цели. В рамках этого процесса используются такие методы, как наблюдение, измерение, опросы, анкетирование, тестирование и т.д.

Представление информации – это преображение информации к форме, наиболее удобной для её использования тому, для кого она предназначена. В рамках этого процесса используются такие методы, как сортировка, систематизация, изменение в табличную или графическую форму и т.д.

Обработка информации – это процесс, при котором изменяется содержание или форма представления информации.

Накопление информации – это процесс создания неструктурированного массива информации по определенному вопросу, пригодного для дальнейшей обработки и анализа.

Хранение информации – это ее запись на носитель (некое запоминающее устройство) для последующей обработки.

Поиск информации – это нахождение нужной информации, заранее известной полностью или только частично, в различных информационных хранилищах (таковыми могут быть каталоги, справочники, поисковые системы и т.д.)

Передача информации – это перемещение информации в пространстве от источника до получателя. В данном процессе переносчиками могут выступать электрические, звуковые, световые волны и т.д.

Использование информации – это обоснованное принятие решений ввиду анализа и оценки информации в разных сферах человеческой деятельности.

Защита информации – это исполнение определенных продуманных мер для предотвращения

утраты, повреждения или злоумышленного использования информации.

Стандартная последовательность информационных процессов: Получение – Хранение – Обработка – Хранение – Передача. Конечно, эта схема изменяется и дополняется в конкретных ситуациях.

1.3. Когда информация находится в безопасности?

Информационная безопасность – это деятельная практика предотвращения несанкционированного доступа, использования, искажения, записи или уничтожения информации. Проще говоря, нужно сделать все возможное, чтобы никакой недоброжелатель не прочел, не воспользовался, не испортил, не подменил и не уничтожил те данные, которыми мы дорожим. Конечно, не всегда всем этим действиям нужно противостоять, следует исходить из конкретной ситуации. Например, если мы говорим об информации на рекламном плакате, то защищать ее от прочтения (доступа) совершенно не нужно, как раз наоборот.

Понятие безопасности универсально и применяется к любым данным вне зависимости от содержания и формы, которые они могут принимать.

Основная задача информационной безопасности – сбалансированная (!) защита информационных свойств, с учётом целесообразности применения и без какого-либо ущерба производительности заинтересованным лицам. Основные свойства в данном вопросе – это целостность, доступность и конфиденциальность (подробнее о них речь пойдет позже).

Баланс же необходимо соблюдать именно для того, чтобы не оголтело бросаться на защиту всего подряд, растрачивая все свои ресурсы, а подходить к вопросу с умом, предварительным проведя анализ и распределив все средства строго по важности задач. Этот баланс достигается, в основном, за счет планомерного многоэтапного процесса анализа и управления рисками, который позволяет распределить основные средства и ресурсы (не только финансовые, но и трудовые, технические и временные) по выявленным источникам угроз и уязвимостям, оценив потенциальную степень воздействия и планируемую тяжесть ущерба.

В рамках данного куса предлагается более упрощенный вариант термина: Информационная безопасность (ИБ) – свойство информации сохранять конфиденциальность, целостность и доступность.

Целостность (Ц) – свойство информации, заключающееся в отсутствии в ней любых изменений за исключением санкционированных.

Доступность (Д) – свойство информации, заключающееся в возможности легального пользователя получить к ней беспрепятственный доступ, возможно, при выполнении условий, установленных владельцем информации.

Конфиденциальность (К) – свойство информации, заключающееся в ее доступности ограниченному кругу лиц, назначенному владельцем информации.

Примеры нарушений свойств: размытые данные в отчете (ц), разбита флешка (д), потеряли черновики разработок (к)

Модель «АБЕ»

При анализе информационной ситуации принято разрабатывать информационную модель, которая формализует и унифицирует конкретные реалии. Специалистам далее удобно проводить анализ, когда конкретика подменена некоторым более стандартизированной моделью. В рамках данного пособия познакомимся с принятой системой Алиса – Боб – Ева, которая представляет упрощенную схему передачи информации между сторонами при наличии нарушителя.

В качестве условных взаимодействующих сторон обозначений принято использовать два имени

Алиса – отправитель и Боб – получатель (как в школьной задаче «из пункта А в пункт Б…»). Важно понимать, что «Алиса», «Боб» и т. п. обозначают не обязательно людей, а вообще агентов, независимо от их реализации: это могут быть, например, компьютерные программы, действующие от имени людей. Алиса и Боб – архетипы именно в области криптографии; Ева – более общее имя, традиционно отданное нарушителю.

Таких имен, принятых в качестве условных обозначений, намного больше, но в общей практике закрепились именно эти три имени. Впервые эти имена были предложены в книге «Прикладная криптография» Брюса Шнайера (Applied Cryptography by Bruce Schneier).

Очень важным вопросом становится следующий: Кто отвечает за защиту самых главных свойств ЦДК в случае необходимости (не стоит забывать, что это небезусловный вопрос и набор свойств, нуждающихся в защите, всегда зависит от конкретной ситуации)? Ответ представлен в двух таблицах ниже

	Алиса (клиент)	Боб
Ц	+	-
Д	-	-
К	+	+

Алиса (сервер)	Боб
+	-
+	-
+	+

Для отправителя существует два варианта набора ответственностей: в случае сервера – отправитель ответственен за все аспекты безопасности информации, а в случае клиента – мы опускаем свойство доступности, так как доступность принято анализировать и защищать только в этом случае. Для получателя информации нет никаких вариантов – он всегда ответственен только за конфиденциальность, если, конечно, ее защита важна в данном конкретном информационном обмене, дело в том, что защита именно этого свойства должна быть в интересах обеих сторон, иначе ничего не выйдет. А вот в случае целостности вся ответственность ложится на отправителя Алису, так как если она не позаботится о сохранности информации перед отправкой, Боб уже ничего не сможет сделать с полученными искаженными данными.

Раздел 2. Компьютерное представление информации

2.1. Системы счисления

Система счисления – это символический метод представления чисел и набор правил для операций над ними. Системы счисления принято разделять на:

- Позиционные – согласно названию, важную роль играет позиция каждого символа в записи, от его места будет зависеть и числовое значение. Эту позицию принято называть разрядом;
- Непозиционные – тут позиция роли не играет, а числовое значение символа всегда постоянно и определяется только по его виду;

Существуют также и смешанные системы счисления, в которых используются оба режима, указанных выше.

Примером непозиционной системы счисления выступит древнеегипетская система счета. В ней каждый символ нес определенное число. Написать их можно было в любом порядке, все равно результат будет вычислен суммированием всех численных значений.

Символ	Значение	Название
	1	Черта
	10	Пятка
	100	Петля
	1 000	Лотос
	10 000	Палец
	100 000	Жаба
	1 000 000	Ра

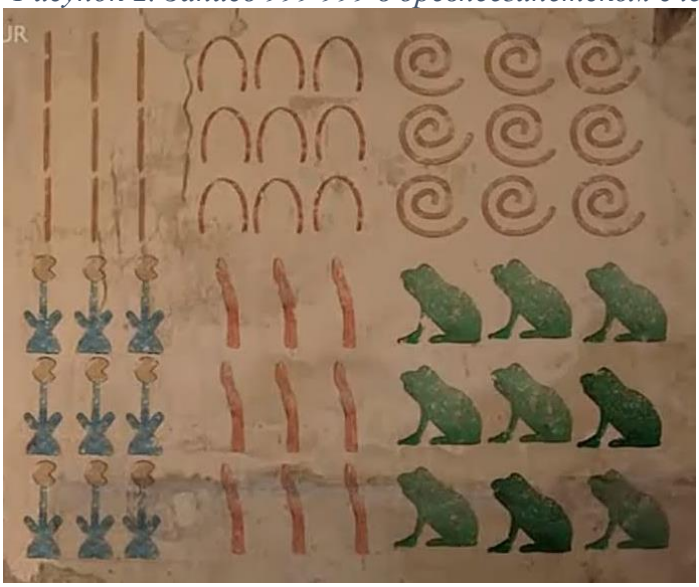
Вертикальная черта означала единицу, чтобы написать, например, три, нужно было провести три вертикальные черты.

Так запись

трактуется как 1 + 1 + 1 + 100 000 + 1 000 + 1 000 + 100 000 + 100 + 1 = 202 104.

Эта система очевидно не экономна, например, чтобы написать 999 999, нужно изобразить соответствующие символы (6 штук) по 9 раз.

Рисунок 1. Запись 999 999 в древнеегипетском счете



Примером почти непозиционной системы является хорошо известная римская система счета. Каждому символу также присущи свои числовые значения, но все-таки они меняют знаки при вычислении результата в зависимости от своего положения. Если перед старшим числом стоит младшее, то младшее нужно вычесть, если же младшее стоит после – его нужно прибавлять.

Есть еще несколько правил записи: на письме символы можно ставить не более трех раз подряд (а V, L, D вообще не могут повторяться), в случае же вычитания из большего числа – не более одного раза, причем только «соседний» по величине символ (то есть, чтобы написать 99 нельзя поставить 1 перед сотней, нужно сначала написать 90, как 10 перед сотней, а затем 9, как один перед десятком, таким образом 99 – XCIX).

Символ	Значение	Название (лат.)
I	1	unus
V	5	quinque
X	10	decem
L	50	quinquaginta
C	100	centum
D	500	quingenti
M	1 000	mille

Например, рассмотрим запись

CDXLII

Здесь используется 100, 500, 10, 50, 1 и 1. Если сначала стоит меньшее число (100 перед 500, и 10 перед 50), то его надо вычесть, а все остальное сложить, таким образом получаем: 500 – 100 + 50 – 10 + 1 + 1 = 442.

Рассмотрим позиционную систему счисления более формально. Под позиционной системой счисления обычно понимается b-ичная система счисления, которая определяется целым числом $b > 1$, называемым основанием системы счисления. Само основание отражает мощность каждого разряда, измеряемую в следующем меньшем разряде. Формально, именно так. Не следует воспринимать основание системы счисления, как «количество символов в записи».

Для примера рассмотрим замечательную 5-ричную систему счисления, в записи которой есть только два символа! Это СКППУ или символы культуры полей погребальных урн. В середине двадцатого века в Германии ученые раскопали 250 серпов, происхождение которых датируется поздним бронзовым веком (примерно, 1500—1250 годов до н. э.). так как признаки бытового использования отсутствовали, было сделано предположение, что назначение у серпов было сугубо ритуальное, тем более что найдены они были в местах погребения. Необычные символы, найденные на них, интерпретировали как числа. Изображения на серпах были следующими: на рукоятях были изображены косые черты вправо и влево, в углу лезвия и на основании были более сложные изображения. Предполагается, что каждая косая черта вправо – это единица, а влево – это пятерка, причем единица не может быть написана более 4 раз подряд. Тут отступает логика разрядов в привычном нам виде, то есть здесь разряд не является позицией, в которую подставляют разные цифры. Символа только два, и «пятерка» приходит на помощь «единице» только, когда единица себя исчерпает, то есть она будет написана максимальное число раз подряд. В отличие от египетской системы записи, тут соблюдался всегда строгий порядок, вероятно продиктованный лишь эстетикой. Интересно также заметить, что максимальное число в таком написании – лишь 29, предположительно, потому что в лунном календаре, используемом в тот период, в каждом месяце не более 29 дней, а на серпах были скорее всего написаны именно даты. Итак, в таблице представлен счет до 29 в записи СКППУ.

Рисунок 2. Система счета культуры полей погребальных урн

/	1	/\	11	/\\\	21
//	2	//\	12	//\\\	22
///	3	///\	13	///\\\	23
////	4	////\	14	////\\\	24
\	5	\\\	15	\\\\	25
/\	6	/\\	16	/\\\\	26
//\	7	//\\	17	//\\\\	27
///\	8	///\\	18	///\\\\	28
////\	9	////\\	19	////\\\\	29
\\	10	\\\\	20		

Чтобы яснее разобраться, какую нагрузку несет это самое основание, рассмотрим привычную нам 10-чную систему счисления. Ее основание, как становится ясно из названия, – 10. Это значит, что каждый разряд может максимально вынести лишь 10 значений, а при желании записать большее число следует переходить в следующий разряд. Эти значения или символы принято называть цифрами, в нашей привычной системе есть десять цифр:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Только это десять значений может нести один разряд, когда нужно написать большее число, мы прибегаем к услугам еще одного разряда – 10.

В позиционных системах чем больше основание системы счисления, тем меньшее количество разрядов (то есть записываемых цифр) требуется при записи числа.

Целое число без знака x в b-ичной системе счисления представляется в виде конечной суммы степеней числа b:

$$x = a_0 \cdot b^0 + a_1 \cdot b^1 + \cdots + a_{n-2} \cdot b^{n-2} + a_{n-1} \cdot b^{n-1} = \sum_{k=0}^{n-1} a_k \cdot b^k,$$

где a_k – это целые числа, называемые привычным нам словом – цифра. Они находятся в интервале значений от 0 до b-1. k в такой записи отвечает за номер разряда, оно пробегает все значения от 0 до $n-1$, а n – это количество использованных разрядов в записи.

Например, рассмотрим в 10-ичной системе счисления число $x = 513$. В данном случае основание $b = 10$. В записи используется три разряда (написаны всего три цифры, значит $n = 3$): разряд сотен ($k = 2$, поскольку сотня получается при возведении основания именно в эту степень: $b^k = 10^2 = 100$), разряд десятков ($k = 1$) и разряд единиц ($k = 0$). Сами разряды заполнены следующим образом: $a_0 = 3, a_1 = 1, a_2 = 5$. В развернутом виде запись такова: $513 = 3 \cdot 10^0 + 1 \cdot 10^1 + 5 \cdot 10^2$

Повсеместно употребляемыми в настоящее время позиционными системами являются следующие:

2 – двоичная (в дискретной математике, информатике, программировании);

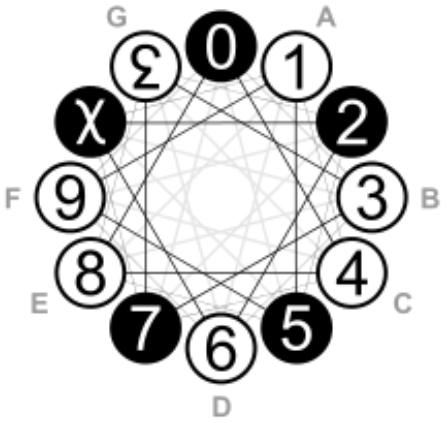
8 – восьмеричная (реже, но все-таки применима в технике);

10 – десятичная (используется повсеместно);
12 – двенадцатеричная (счёт дюжинами);
16 – шестнадцатеричная (используется в программировании, информатике);
60 – шестидесятеричная (единицы измерения времени, измерение углов и, в частности, координат, долготы и широты).

Однако в разных странах приняты свои национальные системы с основанием, отличным от нашей привычной 10. Например, двадцатеричная система принята в некоторых языках Африки (например, в языках йоруба, банту, манде, Того и нигерийских языках), встречается часто в кельтских языках (валлийский, ирландский, шотландский). В Америке основание 20 использовалось в культуре майя и ацтеков. В сантали, одном из официальных языков Индии, используется десятерично-двадцатеричная система при счёте от 19 до 399. В датском языке 20 используется как основание для счёта от 50 до 99. Во французском языке во многих странах 20 используется как основание счёта от 80 до 99.

Двадцатеричная система распространена в кавказских языках. В нахско-дагестанских языках принята двадцатеричная система - у ингушей, чеченцев, аварцев, лезгин. В грузинском 11 произносится как «10-1−больше», дальше; счёт идёт по двадцаткам: 20 – «otsi», 40 – «ormotsi» (что значит «дважды 20»), 50 – «ormotsdaati» («дважды двадцать и десять»), и так далее. А, например, 97 – «четырежды двадцать и семнадцать». В абхазском языке такая же система счета: 20 – «ҩажәа», 30 – «ҩажәижәаба» («двадцать десять»), 40 – «ҩынҩажәа» («два раза двадцать»), 78 – «хынҩажәижәаба жәаа» («три раза двадцать восемнадцать»).

Восьмеричная система чаще всего используется в областях, связанных с цифровыми устройствами. Для нее характерен легкий перевод чисел в двоичные аналоги и обратно, для этого достаточно заменить каждую восьмеричную цифру на ее двоичный триплет (набор из трех двоичных цифр). Широко использовалась в программировании и компьютерной документации, однако позднее была почти полностью вытеснена шестнадцатеричной. Однако не только в компьютере эта система популярна. Она закреплена и в цустном счете южноамериканских индейских племен Юки и Паме, которые используют в быту восьмеричную систему счисления.



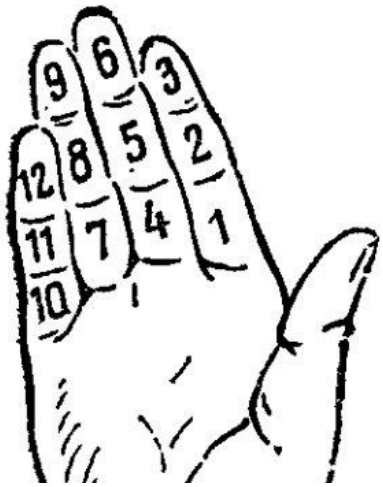


Рисунок 3. Двенадцатеричный циферблат DSA

Рассмотрим еще одну весьма популярную и, пожалуй, самую обсуждаемую, систему счисления. Речь идет о двенадцатеричной системе. Уже не раз по ходу истории возникало мнение, что двенадцатеричная система удобнее и практичнее в быту. Как известно, это мнение до сегодняшних дней не поддержано большинством и отказ от нашей десятичной системы так и не состоялся. Хотя самая успешная известная истории попытка произошла во Франции в XVIII веке под влиянием популярного на тот момент математика и естествоиспытателя – графа де Бюффона. Великая французская революция – период высказывания смелых идей во многих сферах человеческой жизни. В этот момент учреждается специальная «Революционная комиссия по весам и мерам», которая и

принимала решение о переходе к новой системе счисления в бытовом счете. Но аргументы противников реформы (среди которых был, например Лагранж) были весомее.

В середине прошлого века, впрочем, к этой идее вернулись уже в других странах. Наиболее крупными сподвижниками идеи дюжинала являются DSA – The Dozenal Society of America – «Американское двенадцатеричное общество», учрежденное в 1944 году, и DSGB – The Dozenal Society of Great Britain – «Английское двенадцатеричное общество», возникшее пятнадцатью годами позже. Несмотря на все старания пропаганды, неизбежная путаница в период перехода к новой системе счета и большие денежные затраты на переорганизацию всей техники останавливают общество на пути к дюжиналу.

Истоки двенадцатеричной системы счисления находятся в древнем Шумере. По предположениям ученых, первые племена, остановившиеся на двенадцати в качестве основания своей системы счета, отталкивались от количества фаланг руки без учета большого пальца, который как раз помогал считать и запоминать необходимое число, не прибегая к загибанию пальцев, как принято среди европейских народов.

Данная система счета не исчезла в наши дни, ее до сих пор используют некоторые народы Нигерии и Тибета. А еще ее отголоски остались в системе мер. В Риме в основном пользуются унциями (1/12) вместо привычных нам десятых. Английский шиллинг содержит 12 пенни (в отличие 100 копеек в рубле), а дюйм – двенадцатая часть фута (когда сантиметр – сотая часть метра) и так далее.

В самом счете используются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B. Есть также и другое обозначение двух старших разрядов: вместо A используют перевёрнутую двойку, которую называют «дек», и вместо B перевёрнутую тройку – «эль». К удобствам двенадцатеричного счисления можно отнести большее (по сравнению с десятичной системой) количество делителей основания 12 - 2, 3, 4, 6. Чтобы назвать числа, большие одного разряда существуют специальные названия (как и в привычной нам десятичной системе счисления: десять десяток это сто):

10 – до
100 – гро
1000 – мо
10 000 – до мо
100 000 – гро мо
…

Итак, двенадцатеричный счет звучит так: один – два – три – четыре – пять – шесть – семь – восемь – девять – дек – эль – до – до один – до два – … – эль дек – эль эль – гро – гро один – …

На практике двенадцатеричная система осталась и в наши дни в измерении времени, хотя и в смешанном виде. Впрочем, случаем использования двенадцатеричной системы в наши дни может быть представлен счёт дюжинами, широко распространенный в торговле и при комплектации товаров, особенно в Великобритании и Америке. Первые три степени числа 12 имеют собственные названия:

1 дюжина = 12 штук
1 гросс = 12 дюжин = 144 штук
1 масса = 12 гроссов = 1728 штук

Самой популярной в наши дни стала двоичная система счисления, благодаря развитию техники ее использующей. Однако, существуют племена Торосских островов, использующих ее в быту. У них есть только две цифры: урату и окоза. Все остальные комбинации они составляют с их помощью.

Не стоит заблуждаться, считая основание системы счисления показателем количества цифр в ней.

Зачастую это так, но есть и исключения. Напрмер, экзотическая система записи счета – символы культуры полей погребальных урн. Это серия символов, интерпретированных как примитивные цифры, обнаружена на жертвенных бронзовых «серпах» раннего периода центральноевропейской культуры полей погребальных урн, около 1200 года до н. э. в записи чисел используется лишь два символа, но сама система является 5-ричной. Но есть и более удивительные системы – негапозиционные. В них основанием является отрицательное число, а как известно, минус три штуки чего угодно не бывает. Арифметика негапозиционных систем совпадает с привычной нам. Применения таким система не найдено, так что можно считать их плодом пытливых математических умов.

2.2.Арифметика систем счисления

В данном разделе собраны правила и примеры перевода и арифметических операций с различными системами счисления. Если этот материал пройден и хорошо усвоен ранее, можно убедиться в своих умениях сразу перейдя в раздел практики.

Для начала рассмотрим вопрос перевода числа из любой предложенной системы счисления в привычную нам десятичную. Это действие основано на развернутой форме записи, которая была продемонстрирована в примере выше.

Алгоритм перевода в 10-ю систему счисления:

1 шаг. Пронумеровать разряды от младшего (влево), начиная с нуля. Для перевода десятичных дробей следует вести отрицательную нумерацию разрядов после запятой (вправо).

2 шаг. Записать развёрнутую форму числа (разложить по разрядам).

3 шаг. Представить все числа, фигурирующие в развёрнутой форме, в 10-й системе счисления (этот пункт касается систем счисления с алфавитом, содержащим символы отличные от привычных нам цифр, чаще всего с основанием больше 10, в которых принято использовать буквы или прочие символы, имеющие аналог в десятичной системе счисления, например, А – это 10 в 16-ричной системе, или эль – это 11 в 12-ричной системе счисления).

4 шаг. Вычислить значение полученного выражения.

Перевод в десятичную систему счисления целых двоичных чисел будет значительно проще, если вспомнить и использовать уже знакомую вам таблицу степеней двойки.

Рассмотрим примеры $42D_{16}$ и $101{,}01_2$. Далее подробно описаны все шаги алгоритма при переводе этих двух примеров. Итак, при переводе $42D_{16}$ получаем следующее:

1 шаг. $4^{\mathbf{2}}2^{\mathbf{1}}D^{\mathbf{0}}_{16}$

2 шаг. $42D_{16} = 4 \cdot 16^2 + 2 \cdot 16^1 + \boldsymbol{D} \cdot 16^0$

3 шаг. $42D_{16} = 4 \cdot 16^2 + 2 \cdot 16^1 + \mathbf{13} \cdot 16^0$

4 шаг. $42D_{16} = \ 1024 + 32 + 13 = 1069_{10}$

Рассмотрим, $101{,}01_2$

1 шаг. $1^{\mathbf{2}}0^{\mathbf{1}}1^{\mathbf{0}}, 0^{\mathbf{-1}}1^{\mathbf{-2}}{}_2$

2 шаг. $101{,}01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$

3 шаг. пропускаем

4 шаг. $101{,}01_2 = 4 + 0 + 1 + 0 + 1/4 = 5{,}25_{10}$

Теперь рассмотри обратную задачу - перевод целых (для начала) чисел из десятичной системы счисления в указанную систему. Для этого нужно последовательно делить данное десятичное число и попутно получаемые целые частные на основание выбранной системы счисления до тех пор, пока частное не обнулится. Запись результата принято производить в обратном порядке (от самого последнего остатка к началу). При наличии дробной части у числа, ее необходимо отделить и

выполнить перевод отдельно.

Итак, алгоритм перевода целой части числа в систему счисления с основанием b:

1 шаг. Разделить остатком предложенное число на b;

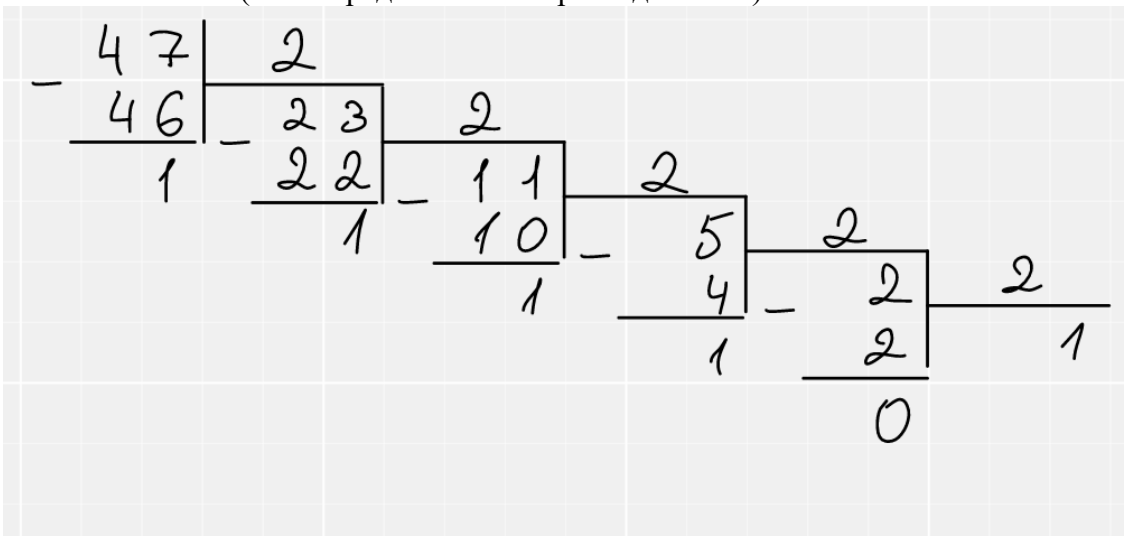
2 шаг. Полученное частное вновь с остатком разделить на b и повторять данный шаг, пока частное не станет меньше b;

3 шаг. Остатки при необходимости записать в соответствии с алфавитом новой b-ичной системы счисления (этот шаг, как и в предыдущем алгоритме, относится к системам с алфавитом, отличным от привычных нам цифр);

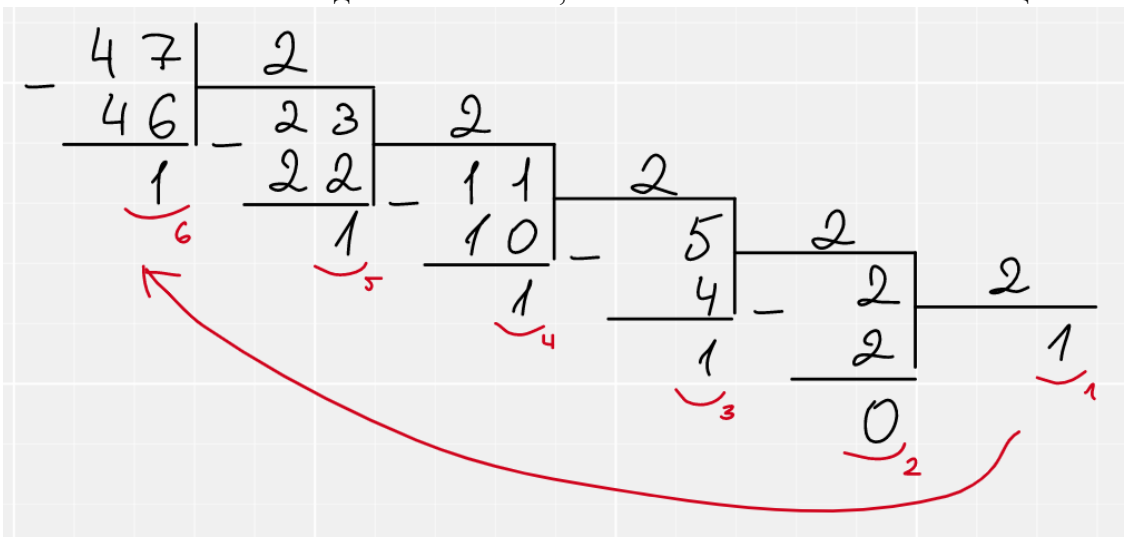
4 шаг. Записать сначала самое последнее частное, а потом все остатки от последнего к первому.

Для примера переведем число 47 из десятичной системы в двоичную, восьмеричную и шестнадцатеричную.

Итак, при переводе в двоичную систему счисления будем последовательно делить число и получаемые частные на 2 (ниже представлены первые два шага):

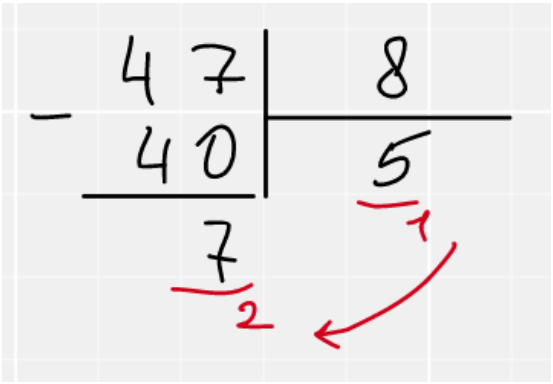


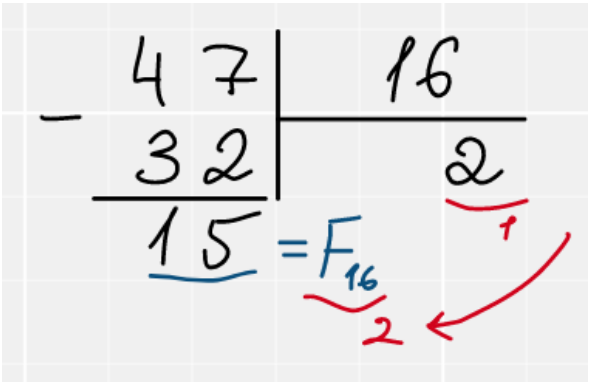
3й шаг опускаем, так как здесь нечего переписывать новыми символами и производим «сборку» (4й шаг): сначала запишем последнее частное – 1, затем все остатки начиная с конца.



В результате получаем запись в двоичном виде: $47_{10} = 101111_2$

Ниже приведены переводы в восьмеричную и шестнадцатеричную системы счисления: $47_{10} = 57_8$, а $47_{10} = 2F_{16}$.





В последнем примере результат третьего шага показан синим цветом.

Отдельно разберем вопрос перевода десятичных дробей из десятичной в новую систему счисления с основанием b. Вместо деления теперь мы также последовательно будем умножать данную дробь на b, пока не получим нужного результата. По аналогии с алгоритмом перевода целой части числа, результатом станут записанные подряд целые части полученных произведений. Окончить процесс можно, если при умножении дробная часть обнулилась. Однако бывает так, что число в новой системе счисления имеет бесконечную (или по крайней мере очень большую) запись, потому зачастую останавливаются в вычислении намного раньше, при достижении определенной точности. При таком подходе возникнет погрешность записи, то есть некоторая неточность, допущение. Предельную абсолютную погрешность (то есть наша ошибка в переводе не будет превышать данного значения) можно оценить по формуле:

$$\frac{1}{2q^{k+1}},$$

где k – количество знаков после запятой в результирующей записи.

Итак, алгоритм перевода десятичной дроби в систему счисления с основанием b:

1 шаг. Умножить предложенную дробь на b, записав отдельно полученную целую и десятичную часть;

2 шаг. Полученную десятичную часть вновь умножить на b и повторять данный шаг, пока не будет достигнут необходимый результат;

3 шаг. Целые части при необходимости записать в соответствии с алфавитом новой b-ичной системы счисления;

4 шаг. Записать по порядку от первой к последней все полученные по ходу вычисления целые части.

Напомним читателю общую рекомендацию: при работе с дробями лучше перевести (или по крайней мере обратить на это внимание мысленно) хотя бы один лишний разряд, чтобы учесть правильное округление.

Для примера переведем число $0{,}47$ из десятичной системы в двоичную, восьмеричную и шестнадцатеричную с точностью – 6 знаков после запятой.

1 шаг. $0{,}47 \cdot 2 = 0{,}94$. Отдельно запоминаем полученную целую часть 0 и далее работаем только с десятичной частью $0{,}94$.

2 шаг. Последовательно продолжаем те же действия: $0{,}94 \cdot 2 = 1{,}88$. Запомнили 1, продолжаем с $0{,}88$.

$0{,}88 \cdot 2 = 1{,}76$. Запомнили 1, продолжаем с $0{,}76$.

$0{,}76 \cdot 2 = 1{,}52$. Запомнили 1, продолжаем с $0{,}52$.

$0{,}52 \cdot 2 = 1{,}04$. Запомнили 1, продолжаем с $0{,}04$.

$0{,}04 \cdot 2 = 0{,}08$. Запомнили 0, а далее продолжать не станем, так как это уже шестое число, которое мы запомнили, значит полученная точность перевода достигнута.

3 шаг. опускаем

4 шаг. Выписываем подряд все целые части: $0{,}47_{10} = 0{,}011110_2$

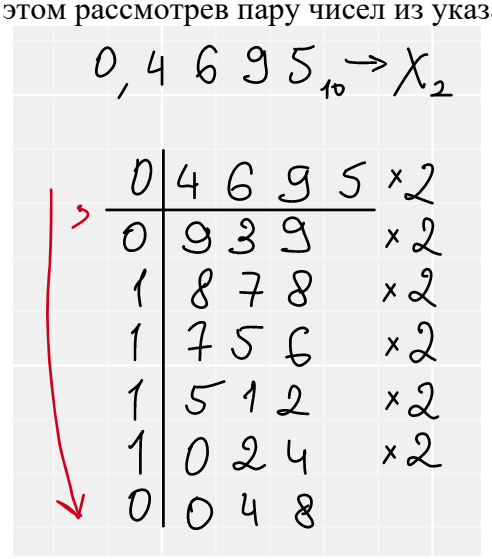
И сразу оценим погрешность перевода, для чего выполним обратный перевод числа и сравним результат с задачей.

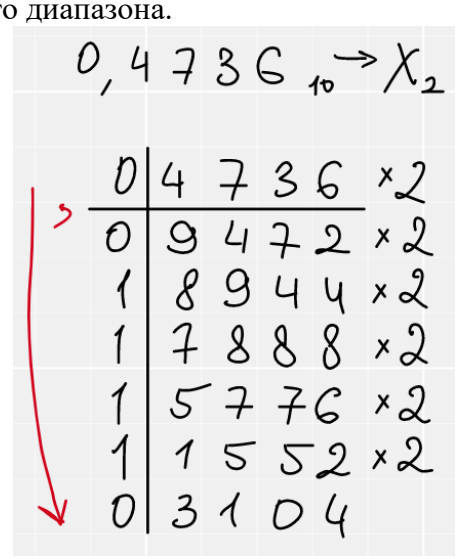
$$0{,}011110_2 = 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} =$$
$$= 0 + 0{,}25 + 0{,}125 + 0{,}0625 + 0{,}03125 = 0{,}46875_{10}$$

Значит, погрешность вычисления составила $0{,}47 - 0{,}46875 = 0{,}00125$.

Вычислим и предельную погрешность: $\frac{1}{2 \cdot 2^{6+1}} = \frac{1}{256} \approx 0{,}0039$. Действительно полученная погрешность вычисления не превосходит предельной.

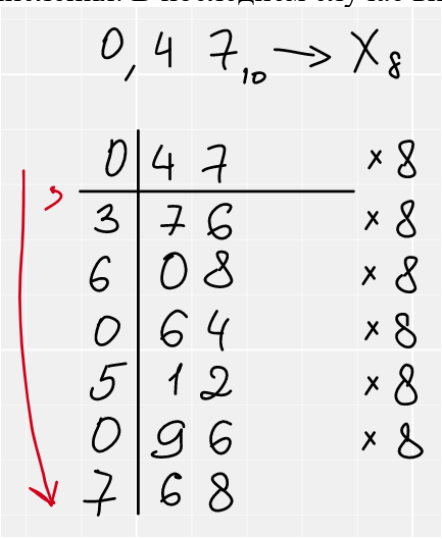
Эта предельная погрешность указывает на диапазон чисел, которые при переводе с той же точностью дадут тот же результат. Таким образом, десятичные дроби в диапазоне $0{,}47 \pm 0{,}0039$ в двоичной системе счисления с точностью в 6 знаков после запятой выглядят как $0{,}011110_2$. Убедимся в этом рассмотрев пару чисел из указанного диапазона.

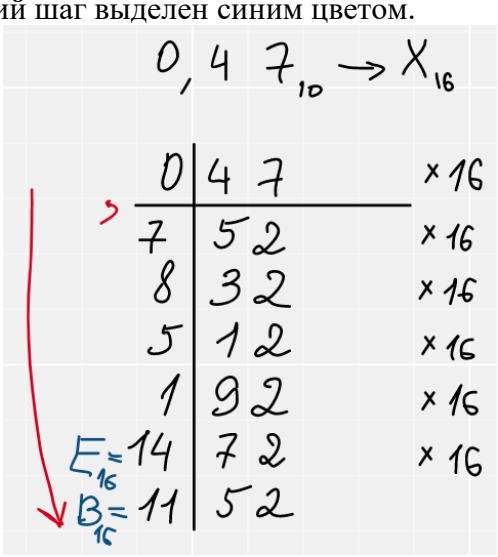




Тут представлена более компактная форма записи перевода. На этапах последовательного умножения вертикальная черта выполняет роль запятой: целая часть записывается слева, а десятичная – справа от нее. Умножается каждый раз только то, что стоит после черты (целая часть игнорируется). На четвертом шаге списывается сверху вниз левый столбик целых частей, причем горизонтальная черта, подводящая заданное число, выступает в роли запятой.

В таком же формате запишем перевод нашей дроби в восьмеричную и шестнадцатеричную системы счисления. В последнем случае вновь третий шаг выделен синим цветом.





Погрешности вычислений рассчитаны ниже (промежуточные вычисления округлены с запасом на один разряд, то есть то седьмого знака):

$$0{,}360507_8 = 0 \cdot 8^0 + 3 \cdot 8^{-1} + 6 \cdot 8^{-2} + 0 \cdot 8^{-3} + 5 \cdot 8^{-4} + 0 \cdot 8^{-5} + 7 \cdot 8^{-6} \approx$$
$$\approx 0 + 0{,}375 + 0{,}09375 + 0{,}0012207 + 0{,}0000267 \approx 0{,}4699974_{10}$$

Значит, погрешность вычисления составляет около $0{,}0000026 = 2{,}6 \cdot 10^{-6}$.

$$0{,}7851\text{EB}_{16} = 0 \cdot 16^0 + 7 \cdot 16^{-1} + 8 \cdot 16^{-2} + 5 \cdot 16^{-3} + 1 \cdot 16^{-4} + 14 \cdot 16^{-5} +$$
$$+11 \cdot 16^{-6} \approx 0 + 0{,}4375 + 0{,}03125 + 0{,}0012207 + 0{,}0000153 +$$
$$+0{,}0000133 + 0{,}0000007 \approx 0{,}47_{10}$$

Здесь получилось заданное число в точности, но это связано с точностью округления промежуточных вычислений (до седьмого знака после запятой). Предельная же погрешность вычислений составляет около $1{,}9 \cdot 10^{-9}$.

В результате получили следующее:

$$0{,}47_{10} = 0{,}011110_2 = 0{,}360507_8 = 0{,}7851\text{EB}_{16}$$

Теперь приступим к изучению правил арифметических операций в позиционных системах счисления. В привычной нам десятичной системе счисления вы хорошо знаете эти правила, не составит трудностей сложить, вычесть, умножить в столбик и разделить уголком числа. В прочих системах идея сохраняется, а меняется только идея с переходом в новый разряд, отчего меняется и таблица умножения.

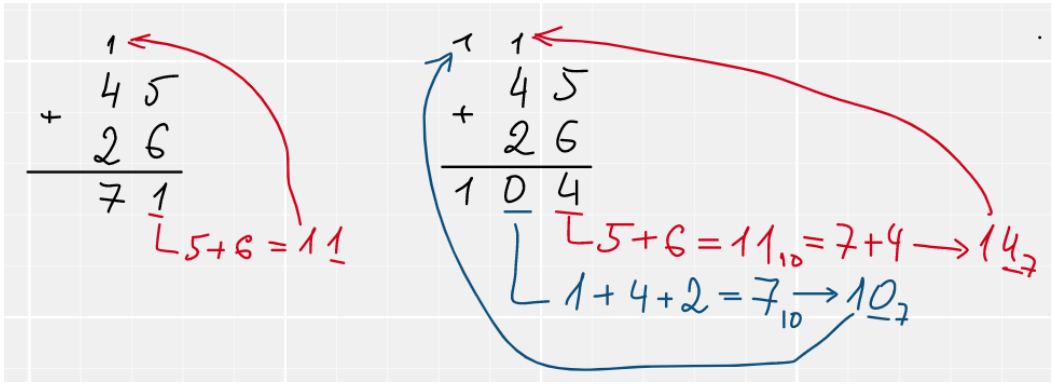
Начнем со сложения. Складывая поразрядно числа, перенос единицы в следующий разряд производим тогда, как сумма достигает основания. Для примера сложим в десятичной и семеричной системах счисления числа 45 и 26. Начиная с младшего разряда, встречаем сумму $5 + 6 = 11$.

В десятичной системе	В семеричной системе
основание 10, результат сложения превысил это число, значит нужно выполнить перенос разряда, оставив в вычисляемом разряде разницу между полученной суммой и основанием, т.е. $11 - 10 = 1$	основание 7, результат сложения превысил это число, значит нужно выполнить перенос разряда, оставив в вычисляемом разряде разницу между полученной суммой и основанием, т.е. $11 - 7 = 4$

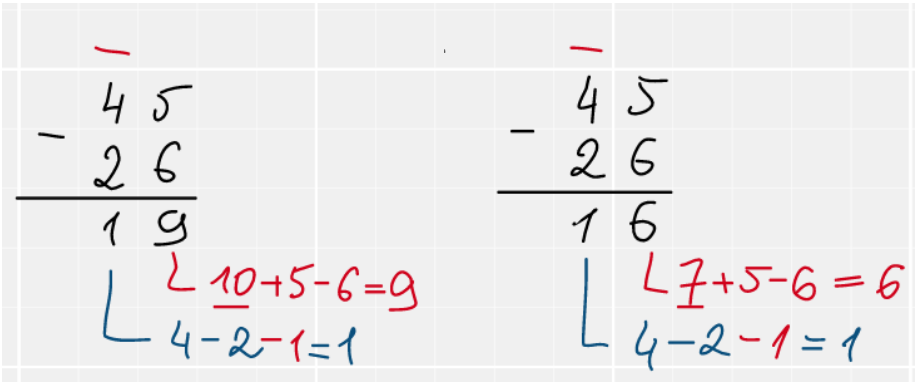
Теперь переходим к следующему разряду (не забыв про перенесенную единицу из младшего разряда): $4 + 2 + 1 = 7$.

В десятичной системе	В семеричной системе
основание 10, результат сложения не превысил это число, значит вычисление разряда окончено.	основание 7, результат сложения достиг это число, значит нужно выполнить перенос разряда, оставив в вычисляемом разряде разницу между суммой и основанием, т.е. $7 - 7 = 0$.
Более разрядов нет, вычисление суммы окончено	В новом разряде есть только перенесенная единица.

Получаем результат: $45_{10} + 26_{10} = 71_{10}$ и $45_7 + 26_7 = 104_7$. Оформим полученные результаты в виде столбика



Основные правила вычитания также не меняются, разница лишь в том, что при заёме из старшего разряда мы получаем не 10, а основание. Оформим также столбиком вычитание 26 из 45 в тех же системах:



Получаем результат: $45_{10} - 26_{10} = 19_{10}$ и $45_7 - 26_7 = 16_7$.

Для умножения в позиционных системах счисления можно заранее подготовить соответствующие таблицы умножения. Основные правила умножения многоразрядных чисел вновь умножения в столбик не меняются: последовательно умножаем одно число на очередную цифру другого множителя и записываем результаты со сдвигом на разряд.

В качестве примера перемножим все те же числа в десятичной и семеричной системах счисления (ниже размещаем таблицу умножения в семеричной системе счислений для справки). В результате получаем:

$$45_{10} \cdot 26_{10} = 1170_{10} \text{ и } 45_7 \cdot 26_7 = 1362_7$$

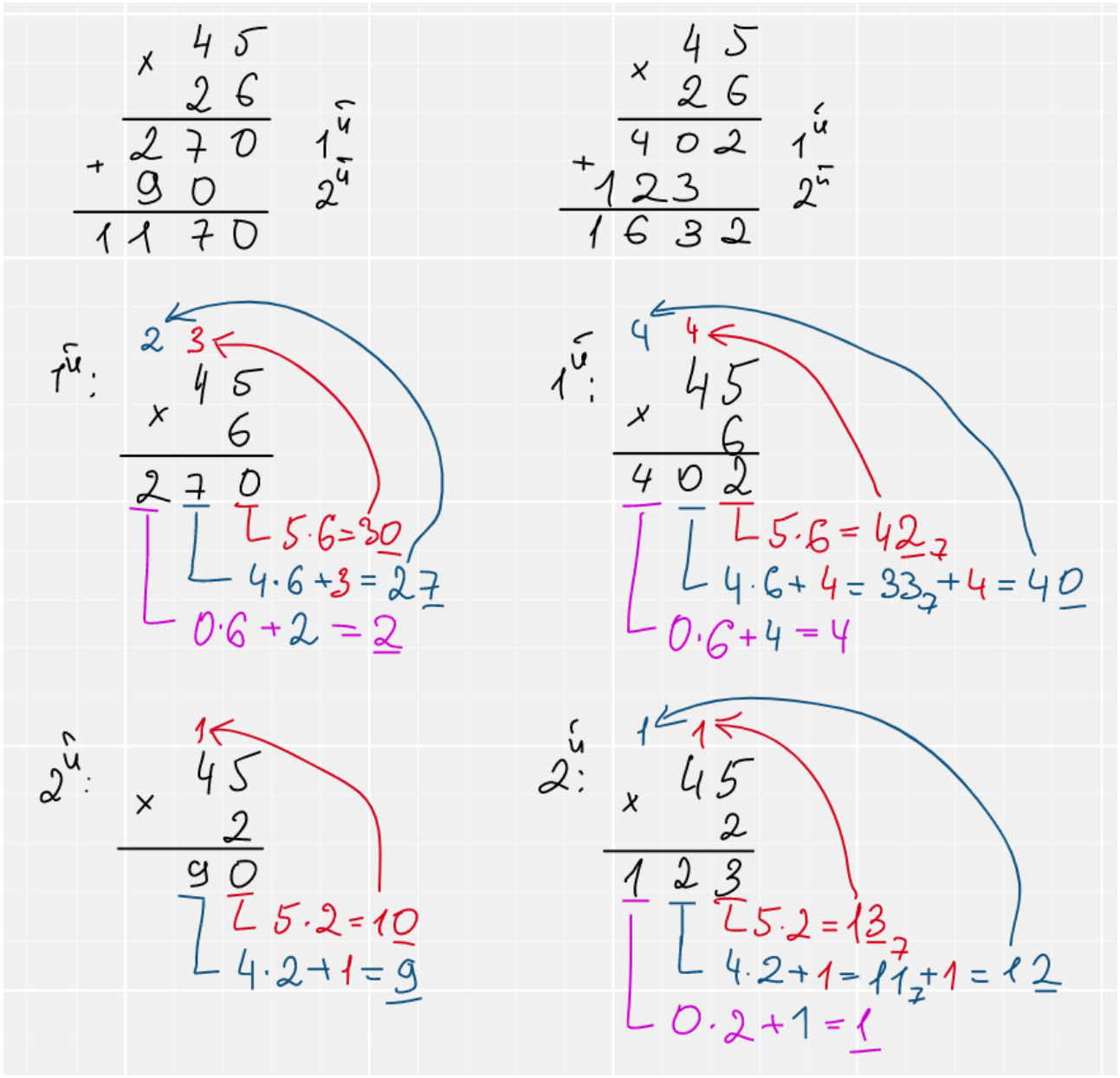


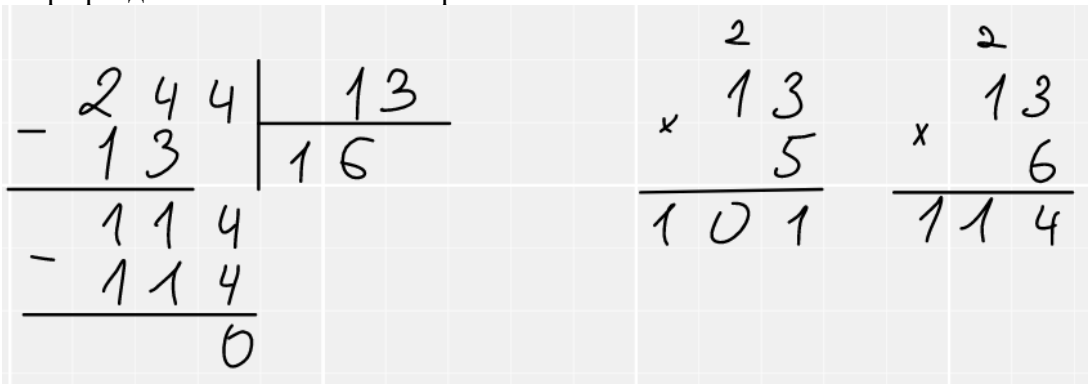
Таблица 1. Таблица умножения в семеричной системе счисления

7СС	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	11	13	15
3	3	6	12	15	21	24
4	4	11	15	22	26	33
5	5	13	21	26	34	42
6	6	15	24	33	42	51

Напоследок осталось деление. Из привычного нам деления уголком в десятичной системе сохраняется идея последовательного подбора разрядов. Тут также пригодится заготовленная

таблица умножения.

Для примера разделим 244 на 13 в семеричной системе счисления.



Значит, $244_7 \cdot 13_7 = 16_7$

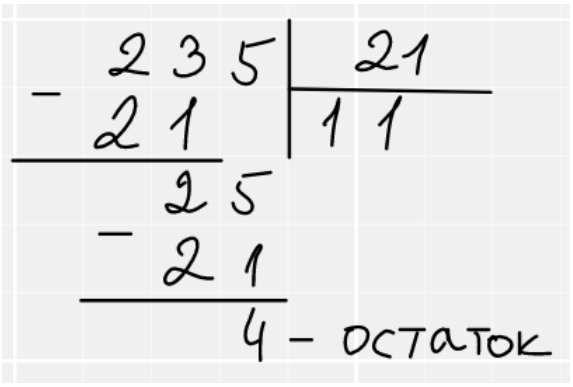
И также выполним деление с уголком, например, разделим 235 на 21 в семеричной системе счисления.

В результате,

$$235_7 : 21_7 = 11_7 \text{ и } 4 \text{ в остатке}$$

2.3.Компьютерное представление чисел

Целые числа представлены в компьютере с двух видах: с учетом знака и в беззнаковом виде. Все хранение происходит в двоичном формате (в двоичной системе счисления).



В зависимости от архитектуры компьютера беззнаковые числа могут занимать в памяти один или два байта. Допустимо заполнение от восьми нулей, что соответствует нулю, до восьми единиц, что отвечает за 255. В расширенном формате (в случае двух байт для хранения одного числа), соответственно, от шестнадцати нулей – так хранится ноль, до шестнадцати единиц – так хранится 65 535. Чтобы узнать самое большое значение беззнакового числа, которое может храниться в предлагаемом объеме памяти, существует формула:

$$2^n - 1,$$

где n – количество предоставленных бит.

Если число хранится в знаковом виде, то самый старший (левый) бит отдается под указание знака. Стандартно принято записывать такие числа в один, два или четыре байта. Диапазон допустимых значений разделен нулем пополам с перекосом в отрицательные числа, то есть из 255 знаковых допустимых значений, которые помещаются в один байт (266-ое значение – это ноль), 128 отрицательных и 127 положительных значений. Соответственно, в двух байтах хранятся значения от – 32 768 до + 32 727.

Далее рассмотрим вопрос хранения чисел с фиксированной запятой. Вообще говоря сама запись таких чисел осуществляется с помощью специальных структур, которым будет посвящено дальнейшее повествование. Существует всего три способа представления двоичных чисел в компьютере:

- Прямой код двоичного числа;
- Обратный код двоичного числа;

– Дополнительный код двоичного числа.

Итак, прямой код используется, в основном, для хранения неотрицательных чисел и представляет собой перевод числа в двоичной системе счисления (при этом, если длина полученного представления не заполняет всю предоставленную память, старшие разряды заполняются нулями). Существует два варианта его использования:

Основной способ – используются все предоставленные биты (то есть, например, для байта их восемь) для целых неотрицательных чисел в десятичном диапазоне от 0 до +255 в байте

Второй способ – используются все предоставленные биты, кроме одного (то есть, например, для байта их семь) для положительных и отрицательных чисел в десятичном диапазоне от -127 до +127 в байте. В этом формате старший (самый левый) бит является знаковым и заполняется нулем для положительных чисел и единицей – для отрицательных. Как бы это ни было странно, в таком формате признаются два нуля: ноль – 00000000 и отрицательный ноль – 10000000.

Этот вариант хранения числовой информации неэффективен, так как для выполнения арифметических операций необходимо выполнить перевод в другое представление или же задействовать для этого центральный процессор, что затруднено.

Обратный код удобно использовать для вычитания с помощью операции сложения с противоположным числом. Если число положительное, то его вид обратного кода совпадает с видом прямого кода, бит знака заполняется нулем. Если число отрицательное, то оно хранится в обратном коде так: бит знака заполнен единицей, остальные биты заполнены инверсией модуля – положительного числа. Инверсия – это процесс замены всех значений каждого бита на другое значение (то есть ноли меняется на единицу, а единица на ноль).

Например, рассмотрим запись в байте 29 и -29. Переведем для начала модуль в двоичный вид: $29 \rightarrow 0001\ 1101$. Так выглядит представление в прямом коде и обратном коде. Теперь рассмотрим противоположное число. В прямом коде старший разряд, отвечающий за знак, должен изменить своей значение, таким образом получим запись: $-29 \rightarrow 1001\ 1101$. В обратном же коде нужно инвертировать все без исключения его биты и получится представление: $-29 \rightarrow 1110\ 0010$. Именно этим и удобны обратные коды в применении – для получения обратного числа достаточно инвертировать все биты. Таким образом, процедура вычитания сводится к двум операциям: чтобы в обратном коде из X вычесть Y, необходимо инвертировать запись Y (получая, таким образом, противоположное число) и сложить полученное с X.

Например, выполним вычитание $35 - 19$ в обратном коде. Имеем

$$35 \rightarrow 0010\ 0011,$$
$$19 \rightarrow 0001\ 0011.$$

Инвертируем запись вычитаемого: $-19 \rightarrow 1110\ 1100$.
Складываем полученный байт и байт 35: $1\ 0000\ 1111$.
Получен новый разряд – лишний девятый бит (это называется переполнением), в таком случае принято поступать так: новый бит сбрасываем, но прибавляем к результату 1, то есть: $0000\ 1111 + 1 = 0001\ 0000$.

Проанализируем результат. Старший бит заполнен нулем, значит полученный результат – положителен (что справедливо, так как вычитаемое не превосходило уменьшаемого). Значит перед нами прямой код и осталось только перевести двоичный код в десятичную систему: $0001\ 0000 \rightarrow 16$. Действительно, $35 - 19 = 16$.

Проведем такую процедуру для такого примера: $11 - 20$.

$$11 \rightarrow 0000\ 1011,$$
$$20 \rightarrow 0001\ 0100.$$

Инвертируем запись вычитаемого: $-20 \rightarrow 1110\ 1011$.
Складываем полученный байт и байт 11: $1111\ 0110$.

Вновь проанализируем полученный результат. Старший бит заполнен единицей, значит полученный результат – отрицателен (что справедливо, так как вычитаемое превосходит уменьшаемое). Значит перед переводом двоичного кода в десятичную систему нужно инвертировать результат, не забыв поставить минус перед полученным числом. Итак, инверсия: $0000\ 1001 \rightarrow 9$. И результат – -9. Действительно, $11 - 20 = -9$.

Из недостатков этого представления чисел – наличие двух нулей и необходимость проводить инверсию в арифметических процессах, поэтому на смену данного формата пришел новый, более продуманный формат – дополнительный код. Именно он наиболее популярен в современных вычислительных машинах. Процедура вычитания также осуществляется в виде сложения, что упрощает работу, унифицируя работу со знаковым и беззнаковым представлением. Это сильно упрощает реализацию вычислительной машины. Здесь вновь старший бит выступает в роли знакового: если знак есть (имеется в виду минус), ставим единицу, если знака нет (то есть, число положительное), то ставим ноль. Тут отсутствует проблема двух нулей и диапазон десятичных чисел в байтовом хранении как раз от -128 до 127.

Запись положительного числа совпадает во всех трех представлениях – это двоичный перевод этого числа. Для представления отрицательных чисел есть два варианта перевода.

Первый вариант состоит из трех шагов:

- Представление в прямом коде модуля предложенного числа;
- Инверсия отрицательного числа, записанного в прямом коде, при этом знаковый бит также меняется, чтобы результат стал отрицательным;
- Прибавление единицы к байту, полученному на предыдущем шаге.

Рассмотрим ля примера число -103:

Перевод модуля в прямой код: $103 \rightarrow 0110\ 0111$.
Инверсия: $1001\ 1000$.
Прибавление единицы: $1001\ 1000 + 1 = 1001\ 1001$.
В результате дополнительный код числа $-103 \rightarrow 1001\ 1001$.

Второй вариант заключается в том, что нужно из нуля вычесть модуль предложенного числа. Фактически, именно эти действия мы выполнили в развернутом виде в предыдущем алгоритме.

Чтобы перевести число из дополнительного кода необходимо выполнить выше предложенный алгоритм в обратном направлении:

- Анализ бита знака: если бит установлен, то число отрицательное, иначе – положительное. Это нужно учесть, записывай ответ;
- Вычитание единицы из предложенного байта;
- Инверсия полученного на предыдущем шаге байта, при этом знаковый бит также меняется – на этом этапе получается модуль прямого кода;
- Перевод двоичного представления в десятичную систему счисления.

Выполним пример на вычитание в дополнительном коде. Рассмотрим, $106 - 103$. Логически необходимо сложить два числа 106 и -103. В дополнительном коде представление положительного числа совпадает с двоичным представлением, а перевод -103 в дополнительный код мы выполнили ранее. Итак,

Первое слагаемое: $106 \rightarrow 0110\ 1010$
Второе слагаемое: $-103 \rightarrow 1001\ 1001$
Сумма: $1\ 0000\ 0011$

В данном случае мы также получаем переполнение, но в отличие от алгоритма в обратном коде тут этот бит принято просто опускать, оставляя результат без изменений. Таким образом, результат: $0000\ 0011$.

Проанализируем ответ. Знаковый бит обнулен, значит, число положительное, что верно математически. В таком случае перед нами двоичное представление числа и остается только перевести его в десятичную систему счисления – это 3.

Рассмотрим еще один пример: $56 - 103$.
Первое слагаемое: $56 \rightarrow 0011\ 1000$
Второе слагаемое: $-103 \rightarrow 1001\ 1001$
Сумма: $1101\ 0001$

Полученный результат является отрицательным числом, судя по установленному знаковому биту. Выполним перевод результата из дополнительного кода:
Вычитании единицы: $1101\ 0001 - 1 = 1101\ 0000$
Инверсия байта: $0010\ 1111$
Перевод в десятичную систему счисления: $0010\ 0000 \rightarrow 47$

В результате получили ответ -47, что действительно является результатом примера $56 - 103$.

Теперь перейдем к рассмотрению вещественных чисел. В математике система вещественных чисел является непрерывной, из чего следует, что в ней бесконечное количество элементов. Безусловно, неограниченное число элементов в памяти компьютера представить не удастся. Поэтому необходимо придумать аналог – дискретную систему (в которой конечное число элементов), для работы вычислительной техники. Такой подход во многом упрощает подход к сравнению и выполнению арифметических операций с вещественными числами. Широко принято использовать экспоненциальную запись или представление числа «с плавающей точкой».

Хорошо известно, что одно и тоже число можно представить в разных записях. Например, $12{,}9 = 1{,}29 \cdot 10 = 0{,}129 \cdot 10^2 = 129 \cdot 10^{-1} = \cdots$

Есть стандартный вид числа – запись в виде произведения мантиссы M на характеристику n^p. Мантисса (в математике) – это число из интервала $1 \leq M < 10$. Характеристика – это степень n^p, где n – основание выбранной системы, p – целое число – порядок. В результате стандартный вид приведенного нами примера именно $1{,}29 \cdot 10^1$, а значит $M = 1{,}29,\ n = 10,\ p = 1$.

Теперь можно уточнить, что это за «плавающая» точка в названии этого представления. Для удобства и унифицирования записи, где бы ни находилось разделение целой и десятичной части числа, мы сместим его в мантиссе в такую позицию, чтобы перед ним осталась только одна значащая цифра (то есть оно не зафиксировано, а переместится). В вычислительной машине принято фиксировать количество разрядов для записи мантиссы, то есть количество значащих цифр числа, что влияет на точность представления. Увеличивая количество бит мантиссы, мы повышаем точность представления вещественного числа. При увеличении количества бит для хранения порядка мы увеличиваем сам диапазон хранимых значений данного формата.

Для компьютерного представления числа понятие мантиссы отличается, в технике – это целое число, состоящее из старших разрядов числа. Для удобства вычислений принято сохранять нормализованный вид – это представление мантиссы, первый бит которой всегда установлен и потому не хранится в памяти, но подразумевается при переводе. Так как вся информация в вычислительной технике хранится в двоичном формате, диапазон для мантиссы нормализованного вида выглядит так: $0{,}1_2 \leq M < 1$. В компьютерном представлении мантисса записана в двоичном представлении, а порядок – в десятичном.

Вычисление числа, представленного в таком виде, существует формула:

$$(-1)^{\text{бит знака}} \cdot 1, M \cdot 2^p$$

Для хранения вещественного числа есть несколько широко распространенных форматов:

– Одинарный формат: 32 разряда – нормализованное знаковое число.

Первый старший бит отдан традиционно под знак. Далее 8 бит хранят порядок (степень, в которую надо возводить двойку) в виде $p + 127$. И замыкает запись мантисса – 23 бита (старший бит мантиссы не хранится, так как в нормализованном виде всегда по умолчанию установлен).

– Двойной формат: 64 разряда – нормализованное знаковое число.

Старший знаковый бит, 11 бит порядка и 52 бита мантиссы (правила записи те же, что и в предыдущем формате).

– Расширенный формат: 80 разрядов – знаковое число.

Запись производится аналогично, старший бит отдается под хранение знака, 15 бит – для хранения порядка, 64 бита – для мантиссы (этот формат позволяет записывать ненормализованные числа).

Рассмотрим пример, как будет записано число $0{,}1041$ в 32-битном формате с плавающей запятой. Для начала переведем это число в двоичную систему счисления (алгоритм был упомянут в предыдущем параграфе), хотя мантисса хранит только 23 бита, перевод следует выполнит с запасом, так как следующим шагом будет выполнено смещение до нормализованного вида:

$$0{,}1041 \rightarrow 0{,}00011010101001100100110000010 \ldots$$

Для нормализованного вида необходимо, чтобы перед запятой стояла только одна единица, для этого нужно сместить запятую на 4 знака вправо, таким образом порядок $p = -4$. Хранится двоичное представление числа $p + 127 = 123 \rightarrow 0111\ 1011$.

Нормализованный вид содержит мантиссу $1{,}10101010011001001100000 10$, старший бит установлен в записи по умолчанию, поэтому его записывать не станем, а после запятой нужно взять 23 бита. Итак, в памяти компьютера для $0{,}1041$ будет записано следующих 32 бита:

$$0 - 0111\ 1011 - 1010\ 1010\ 0110\ 0100\ 1100\ 001$$

Рассмотрим обратный пример, допустим в памяти компьютера хранится такие 4 байта: $1011\ 1110\ \ 0101\ 0001\ \ 0000\ 0000\ \ 0000\ 0000$

Для начала разобьем его на составляющие: бит знака – порядок – мантисса:

$$1 - 0111\ 1100 - 1010\ 0010\ 0000\ 0000\ 0000\ 000$$

Бит знака установлен, значит число отрицательное.

Высчитаем смещение: $0111\ 1100_2 \rightarrow 124_{10}$. Смещение $= 124 - 127 = -3$.

Мантиссу надо сместить на 3 знака влево, не забыв, что по умолчанию в сохраненном формате указаны числа после запятой, а перед ней – единица, таким образом:

$$1{,}1010\ 0010\ 0000\ 0000\ 0000\ 000 \rightarrow 0{,}0011010\ 0010\ 0000\ 0000\ 0000\ 000$$

Осталось перевести полученную двоичную запись в десятичную систему счисления:

$$2^{-3} + 2^{-4} + 2^{-6} + 2^{-10} = 0{,}2041015625$$

Можно было опустить такое явное смещение, но не забыв учесть его в вычислениях таким образом:

$$1{,}1010\ 0010\ 0000\ 0000\ 0000\ 000 = 2^0 + 2^{-1} + 2^{-3} + 2^{-7}$$

С учетом смещения на три получим как раз: $2^{0-3} + 2^{-1-3} + 2^{-3-3} + 2^{-7-3}$

Безусловно, используя такой подход можно хранить и целые числа в точности, если количество бит для его хранения не превышает размер блока, отданного под мантиссу.

Подробнее арифметику вещественных чисел разбирать в этом курсе мы не станем. Оставим лишь несколько комментариев о подготовительных процессах и действиях.

Перед выполнением сложения или вычитания принято «готовить» число с помощью операции выравнивания порядков (как делают и в привычной математике). Это происходит за счет смещения мантиссы числа с меньшим порядком вправо на количество разрядов, позволяющее приравнять порядки (то есть смещаем на разность порядков этих чисел). После такой процедуры соответствующие разряды чисел в мантиссе оказываются на одних и тех же позициях и арифметические операции происходят прямым действием. При необходимости итоговый результат вновь нормализуется.

Для умножения двух вещественных нормализованных чисел нет необходимости выравнивать порядки, их достаточно слоить, а мантиссы перемножить. Для деления порядки двух вещественных нормализованных чисел вычитаются, а мантиссы делятся. При необходимости итоговый результат также нормализуется.

Занимательным также является факт особо оговоренной записи для такого числа, как ноль. С общепринятой традицией хранить нормализованную запись с неявной единицей по умолчанию, появилась проблема нуля – ноль нельзя записать в таком нормализованном виде. Поэтому для его обозначения придумали такие правила: порядок хранит значение -1 (то есть в записи компьютера находится $0111\ 1110$), а мантисса заполняется нулями. Конечно, арифметические операции с нулем при таком подходе необходимо обрабатывать специальным образом, однако на практике они выполняются даже быстрее, чем операции с прочими вещественными числами. Кстати, если мантиссу заполнить нулями, а в мантиссу записать 1 (то есть $1000\ 0010$), то эта запись будет хранить бесконечность (бит знака используется по назначению). Именно наличие этих двух бесконечностей (чего не было в предыдущих форматах записи) придает смысл двум нулям (положительному и отрицательному), так что в данном формате их оставили умышленно.

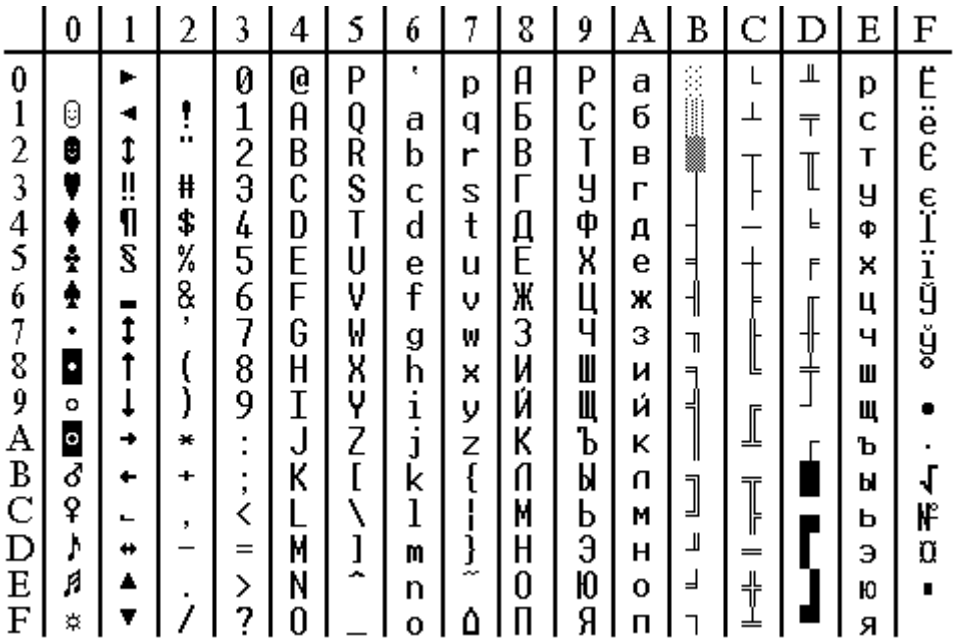
2.4.Компьютерное представление текстовой информации

В данном параграфе мы рассмотрим вопросы хранения текстовой информации. Идея с переводом информации в двоичный код сохраняется, но теперь нельзя просто воспользоваться некими математическими операциями, как мы делали с числами. Вместо математического расчета принято использовать метод сопоставления с эталоном. Каждый символ на нашей клавиатуре в представлении компьютера получил себе некое число. при нажатии любой клавиши, отвечающей за букву, цифру или служебный символ, мы посылаем сигнал в виде двоичного кода, соответствующего этой цифре. Все соответствия Символ – Число хранятся в особой таблице, которая называется кодовой. Кода компьютер получает задачу вывода символа (это также сигнал двоичного представления), происходит декодирование и на основе такой таблицы строится изображение символа.

Существует несколько стандартов такого сопоставления. С середины XX века получила широкое распространение кодировка ASCII (American Standard Code for Informational Interchange – Американский стандартный код информационного обмена). В ней под запись одного символа отводится один байт (что соответствует восьми битам, в которых можно расположить, как известно, 256 двоичных кодовых комбинаций). Хотя изначально разработчики разместили все символы только в семь бит (задействую 128 записей). Поскольку общепринято было использование байтовой системы (то есть минимальной единицей памяти был именно байт, то есть восемь разрядов), семибитные символы расширялись добавлением старшего бита. Сперва его использовали как контрольный бит для проверки ошибок передачи сигнала. Затем таблицу переформатировали и все восемь бит стали значащими.

Общепринято записывать не восемь разрядов двоичного кода, а только два разряда кода шестнадцатеричного, что короче и проще. Для примера, рассмотрим латинскую букву A. В ASCII коде ей соответствует цифра 65_{10}. Таким образом сигнал компьютера, отвечающий за данную букву, это – $0100\ 0001$, но для удобства записи мы будем иметь в виду под ней 41_{16} в таблице (напомним, что для быстрого перевода шестнадцатеричного числа в двоичный код достаточно каждую его цифру заменить на четыре разряда ее двоичного разложения, то есть $4_{16} \rightarrow 0100_2$, $1_{16} \rightarrow 0001_2$). Ниже представлена полная ASCII таблица (верхняя строка отвечает за разряд десятков, а самый левый столбик – за разряд единиц в кодовом числе символа).

Рисунок 4. Кодовая таблица ASCII



Разберем тело ASCII таблицы подробнее. Первые 32 символа (первые два столбика) заполнены так называемыми управляющими символами. Графические картинки нужны лишь для визуального отличия разных байт, но сами эти двоичные последовательности играют роль команд управления. Что они делают непосредственно, зависит от вычислительной техники, на которой ведется их обработка. Изначально эта таблица разрабатывалась для передачи информации по телетайпу (печатная машина, которая позволяла пересылать текстовые сообщения по электрическому каналу связи). И эти 32 символа были не печатными, использовались для указания структуры сообщения (чтобы отделить заголовок сообщения, в котором указывались адреса отправителя и получателя, различные специальные элементы контроля сообщения, от самого текста), запросов подтверждения получения, ответов на такие запросы, указание ошибки и так далее.

Следующие 32 символа таблицы ASCII отданы под хранения служебных символов (знаки пунктуации, стандартные математические обозначения арифметических операций и прочее) и десятичных цифр. Далее идут заглавные латинские буквы и маленькие латинские буквы. Вторая половина ASCII таблицы отдана стандартом под хранение символов национального алфавита и прочих символов.

Хотя кодировка ASCII в наши дни все еще используется в ряде технических средств, большее распространение, особенно в сети интернет, получила новая кодовая таблица – Unicode. В 1991 году этот стандарт был разработан Международной организацией по стандартизации (ISO). Новый стандарт позволял хранить в единой кодовой таблице все необходимые символы разных народных письменностей. Восьмибитных кодировок к концу XX века появилось в огромном количестве, потому как многие страны стали проявлять свой интерес к вычислительной технике. Такое многообразие приводило к закономерным проблемам. Из-за отсутствия стандартизированного способа указания кодировки для сообщения, в сообщении не улавливались символы иностранных языков. Из-за ограниченности набора допустимых символов часто сталкивались с проблемой переноса документа на другую систему (все нестандартные символы, специальным образом прописанные в одной системе, не улавливались в новых условиях). Существовала и проблема перекодировки, чтобы перевести сообщение в другой формат необходимы были громоздкие таблицы соответствий. Также эти таблицы соответствий помогли бы решить проблему дублирования, поскольку даже совпадающие символы в разных кодировках указывались своим написанием. В результате анализа ситуации было решено создать единую универсальную расширенную кодировку.

Для расширения пространства записи выбрали двухкратное увеличение и первая версия Unicode взяла для одного символа 16 бит, сто позволяло хранить уже $2^{16} = 65\,536$ записей. Каждый символ

получал в соответствие теперь число из четырех шестнадцатеричных разрядов в диапазоне от 0000 до FFFF. Коды в стандарте Unicode разделены на семнадцать условных плоскостей (плоскость – это непрерывный диапазон кодов от 0000 до FFFF, который обозначается шестнадцатеричным числом от 0 до 10):

U+0 0000…U+0 FFFF – (нулевая) Базовая многоязыковая плоскость содержит символы более ходовых в наши дни письменностей. Диапазон от 0000 до 00FF отдан под стандарт ASCII. Далее каждой письменности выделен свой блок кодов, сохранены европейские письменности нелатинского происхождения, письменности Африки, Среднего Востока и Азии, Индонезии и Океании и прочие символы.

Остальные плоскости дополнительные:

U+1 0000…U+1 FFFF – (первая) Дополнительная многоязыковая плоскость используется в основном для исторических письменностей и условных обозначений (например, в диапазоне 1D100 – 1D1FF хранятся музыкальные символы, а в диапазоне 1FA00 – 1FA6F – шахматные).

U+2 0000…U+2 FFFF – (вторая) Дополнительная идеографическая плоскость, которая хранит редко используемые иероглифы китайского письма.

U+3 0000…U+3 FFFF – (третья) Третичная идеографическая плоскость зарезервирована под устаревшие китайские иероглифы.

U+4 0000…U+D FFFF – области 4 – 13 пока не используются.

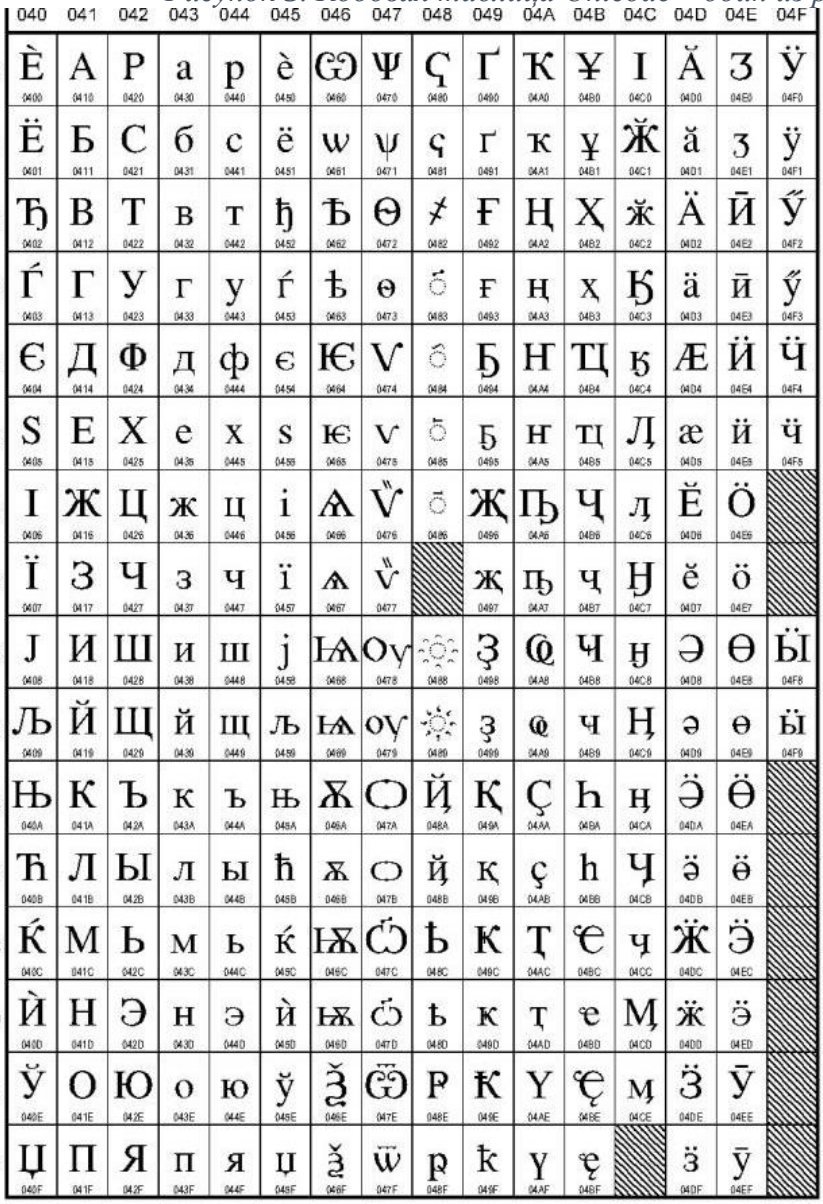
U+E 0000…U+E FFFF – (14-я) Дополнительная специализированная плоскость символов особого назначения (это тэги и селекторы), а большая ее часть еще не заполнена.

U+F 0000…U+10 FFFF – (15-я и 16-я) Дополнительные области для частного использования и экспериментов.

Также разработано несколько форм представления, в зависимости от размера блока. Помимо стандартного двухбайтного UTF-16, есть и компактный однобайтный UTF-8 и расширенный UTF-32 (который, благодаря своей организации прямой индексации позволяет колоссально экономить время в процедурах поиска, хотя и платимся за это большой растратой памяти, что зачастую неоправдано). Все эти способы представления, конечно, взаимно совместимы.

До недавнего времени в этом стандарте были закодированы только все живые официальные письменности (хотя в мире насчитывается 7174 живых языка на сегодняшний день, официальными языками государства являются только около 150 из них). Из всех возможных 65 536 кодовых позиций не было заполнено и половины. Но этот процесс не прекращается и сейчас разработчики стандарта кодируют остальные письменности (мертвые и искусственные языки). Например, в Unicode закодированы символы наших давних славянских предков: символы кириллической письменности (диапазоны от 0400 до 052F, от 1C80 до 1C8F, от 2DE0 до 2DFF,от A640 до A69F в базовой плоскости) и символы глаголицы (диапазоны от 2C00 до 2C5F в базовой плоскости, от 1 E000 до 1 E02F в первой многоязыковой плоскости).

Рисунок 5. Кодовая таблица Unicode – один из разделов кириллицы



2.5.Компьютерное представление графической информации

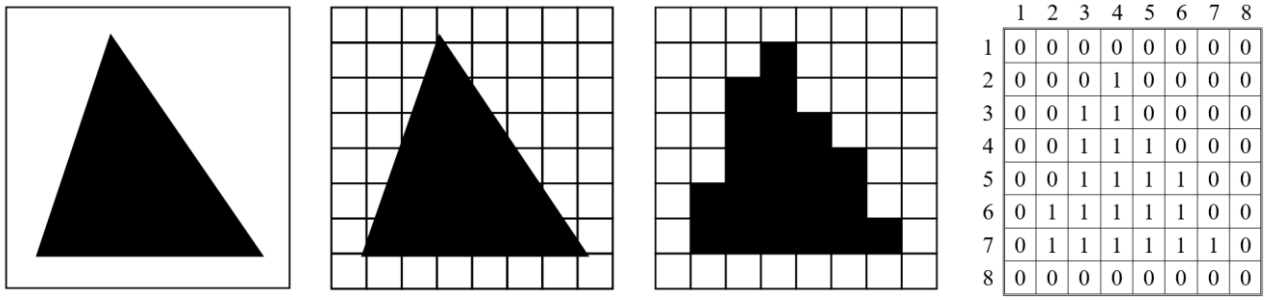
Теперь коснёмся вопросов представления графики. Когда необходимых для запоминания элементов счетное количество (буквы, цифры, прочие символы) легко создать таблицу соответствия их определенным подовым последовательностям. Но изображений, конечно, огромное множество, всех размеров, цветов и содержания. Тут придется искать в них определенную структуру и базовые элементы, которые и будут кодироваться двоичным представлением в вычислительной технике. Таких подходов к описанию структуры изображения два: растровый и векторный.

С точки зрения растрового формата записи изображение – это набор светящихся точек, которые называются пикселями. Все изображение разбивается сеткой на квадратики (количество этих квадратиков называют размером в пикселях, и оно напрямую влияет на качество того, что мы видим). Далее у каждого квадратика определяется цвет (безусловно, чем больше цветов используется в палитре, тем лучше и красочнее будет результат). Например, если нужно сохранить черно-белое изображение, для хранение каждого составляющего пикселя потребуется ровно один бит, так в памяти будет сохранено либо значение 1, что значит черный, либо значение о, что значит белый (формально $2^0 < 2 \leq 2^1$). Если необходимо записать картинку, использую 10 цветов, то минимально потребуется 4 бита для хранения каждого пикселя, так как $2^3 < 10 \leq 2^4$. Эта

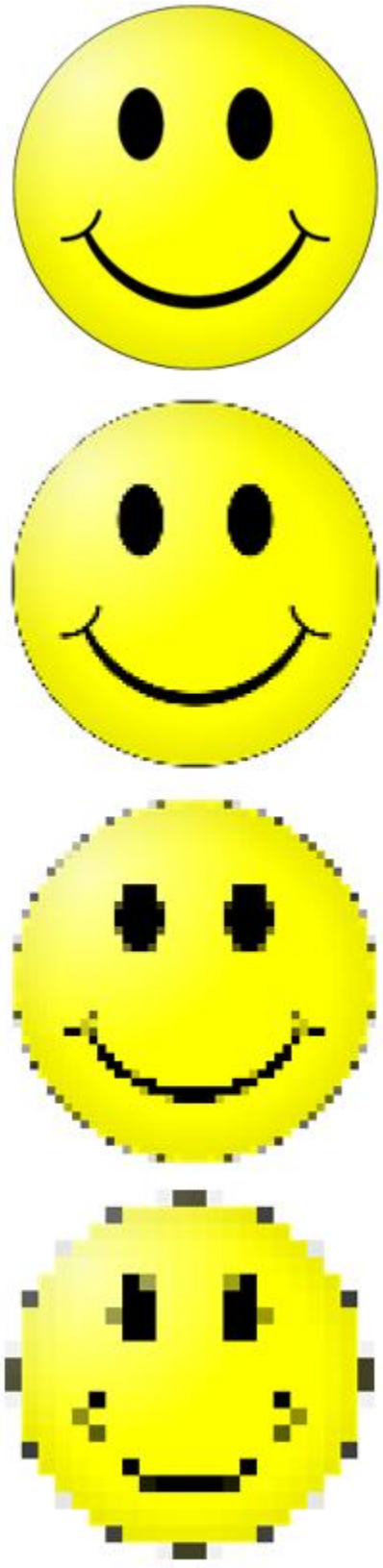
характеристика получила название глубина цвета (количество бит для кодирование одного пикселя). Она определяется по формуле Хартли, то есть минимальной степенью двойки, такой, чтобы необходимое число цветов ее не превосходило.

Посмотрим, как размер влияет на качество. Допустим, есть черно-белое изображение треугольника и необходимо его сохранить в памяти имея только поле 8×8. В таком размере очертания треугольника уже слабо уловимы. Обратите внимание, что самый первый эталонный треугольник так же представим в пикселях, но их значительно больше, поэтому картинка выглядит такой гладкой.

Рисунок 6. Пример кодирования черно-белого растрового рисунка



	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	0	1	1	1	0	0	0
5	0	0	1	1	1	1	0	0
6	0	1	1	1	1	1	0	0
7	0	1	1	1	1	1	1	0
8	0	0	0	0	0	0	0	0



В левой колонке вашему вниманию предлагается изображение смайлика в четырех вариантах размера в пикселях.

Как не трудно догадаться, качественное красивое изображение требует больше памяти для хранения, так как будет записана информация о большом количестве элементов и каждый элемент требует для записи больше бит памяти (чем больше глубина, тем больше бит нужно).

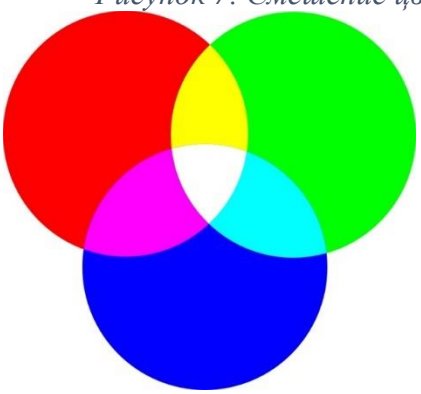
Существуют еще такие характеристики как цветовое пространство или цветовая модель и разрешение изображения.

Разрешение не следует путать с размером (что происходит зачастую). Эта характеристика определяет количество пикселей, помещаемых на единицу площади (квадратный сантиметр, например). Если мы растягиваем или сжимаем картинку, то количество пикселей в ней не меняется, но меняется размер самого пикселя. Это и есть разрешение.

Цветовая модель – математическая модель описания представления цветов в виде цветовых координат (наборов из трех или четырех чисел). Учеными установлено, что наше зрение трихроматическое, то есть сетчатка глаза здорового человека имеет всего три вида рецепторов (так называемых колбочек), ответственных за цветовое восприятие: L-колбочки (long wavelength), M-колбочки (middle wavelength) и S-колбочки (short wavelength). Название они получили от длин волн цветового видимого спектра, на которые они реагируют.

Одна из самых популярных цветовых моделей на сегодняшний день – это RGB (red, green, blue). Основными цветами в ней выступают красный, зелёный и синий. Все они в разных пропорциях добавляются к черному. Если не добавить ничего – пиксель будет черным, если добавить все три в равной пропорции – будет белый. Другие сочетания будут давать прочие оттенки.

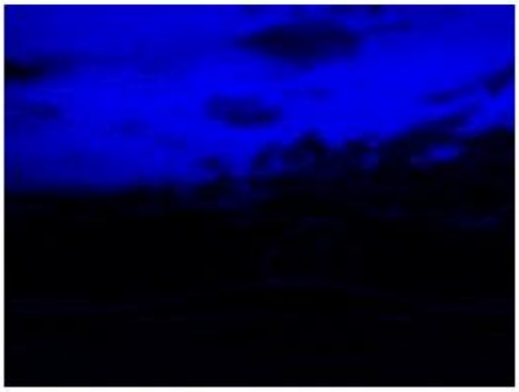
Рисунок 7. Смешение цветов RGB-модели



Раньше в старых «пузатых» телевизорах использовались три электронные пушки, которые подсвечивали соответствующий пиксель согласно его коду, сейчас в тонких жидкокристаллических и матричных мониторах расположены специальные светодиоды, которые переносят все те же три цвета.

Итак, в записи компьютера про каждый пиксель нам нужно указать его цвет (как в случае с черно-белым изображением, указывать только наличие цвета не выйдет, теперь нужно указывать и пропорции основных цветов для достижения нужного оттенка). Визуально принято представлять модель RGB в виде цветового куба, таким образом мы получаем координаты, разложенные на осях.

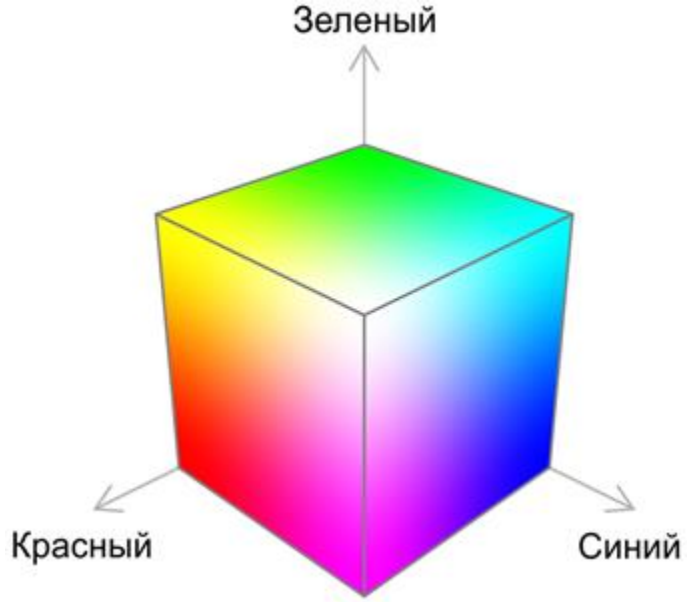
Куб выбирают единичным, то есть $(1; 0; 0)$ – красный цвет, $(0; 1; 0)$ – синий цвет, а $(0; 0; 1)$ –



зеленый цвет. Если нужен белый, то берем все цвета по максимуму $-(1;1;1)$, черный – не берем ничего, то есть $(0;0;0)$. У оттенков указываются соответственно координаты этого куба в виде трех десятичных дробей.

В левой колонке приведены четыре изображения, которые соответствуют оригинальной картинке и ее трем изображениям-каналам (красному, зеленому и синему соответственно).

Рисунок 8. Цветовой куб RGB



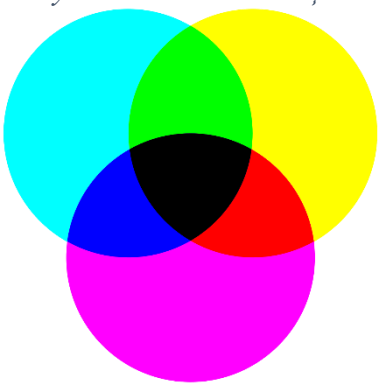
При хранении самого изображений, про каждый в отдельности пиксель не пишут три подряд координаты, на самом деле такие изображения хранятся в виде трех отдельных изображений-каналов.

На практике на самом деле, учитывая физиологические особенности глаза, под кодирование синего и красного каналов отводится по пять бит на пиксель, а на кодирование зеленого – шесть бит на пиксель (потому что наш глаз более восприимчив к зеленым оттенкам). Или же упрощенно все каналы кодируются пятибитно. Отводить под каждый канал целый байт, неэкономично, хотя и можно будет закодировать около 16 миллионов цветов, наш глаз столько различить все равно не сможет.

Существуют и другие цветовые модели, например, CMY (cyan, magenta, yellow) – антипод модели RGB – субтрактивная схема кодирования цвета, в основе которой лежат три цвета: голубой, пурпурный и жёлтый. Модель RGB называется аддитивной (от лат. additivus – «прибавляемый»), потому что для получения нужного оттенка базовые цвета добавляются к черному. Модель CMY напротив считается субтрактивной (от лат. subtragerus – «вычитаемый»), потому что нужный оттенок достигается за счет вычитания соответствующих базовых цветов. Дело в том, что желтый объект не отражает синих волн, а значит можно сказать, что жёлтый вычитает из отражённого белого света синий. Аналогично происходит с красным – это белый минус голубой, и зеленым – белый минус пурпурный. В данной модели белый получается, наоборот, отсутствием каких-либо примесей, а черный – воздействием всех (если из белого вычесть

все цвета, получится черный). Правда, при таком подходе он получится более блеклым, чем настоящий черный цвет. Поэтому данную модель принято расширять до CMYK, добавляя к трём цветам чёрный.

Рисунок 9. Смешение цветов CMY-модели



Чтобы вычислить информационный объём растрового изображения прибегают к формуле:

$$V = N \cdot q,$$

где N – размер в пикселях, q – глубина в битах.

Например, рассмотрим картинку 10×10 пикселей. Всего их 100. Если картинка черно-белая, то для хранения двух цветов, возможных для одного пикселя, нам хватит одного бита. Получается информационный объем такого изображения $V = 100 \cdot 1 = 100$ бит. Если между чёрным и белым цветами мы станем учитывать ещё шесть оттенков серого цвета, то всего в нашей палитре будет уже 8 цветов, в таком случае глубина цвета уже 3 (так как $2^2 < 8 \leq 2^3$). В данном случае информационный объем такого изображения $V = 100 \cdot 3 = 300$ бит. Если мы будем сохранять картинку с шестнадцатью различимыми цветами, то глубина цвета будет уже 4 бита на пиксель (так как $2^3 < 16 \leq 2^4$) и информационный объем такого изображения составит уже $V = 100 \cdot 4 = 400$ бит.

В левой колонке можно увидеть, как количество цветов в палитре влияет на качество изображения (первое изображение учитывает 256 цветов, второе – 32, третье – 16, четвертое – 8 цветов).

Общепринято обозначать цветовые координаты в формате трех пар шестнадцатеричных чисел, каждая пара отвечает за свой цвет – красный, зеленый и синий соответственно. Чтобы обозначить, что такой блок отвечает за цвет, вначале ставят специальный символ – решетка. Ниже указаны шестнадцатеричные коды некоторых цветов:

#000000 – черный цвет (отсутствие цвета);
#FF0000 – интенсивно красный цвет (максимум красного, больше ничего);
#00FF00 – интенсивно зелёный цвет;
#0000FF – интенсивно синий цвет;
#FFFFFF – белый цвет (все компоненты на максимум);

#FFFF00 – интенсивно желтый цвет (смещение красного и зеленого);
#FF5555 – светло-красный цвет (основной цвет – на максимум, остальные увеличены в равных пропорциях);
#808080 – серый цвет (все три равные компоненты дают серый цвет разной степени интенсивности).

Хотя растровая графика имеет много преимуществ и более распространена в наши дни, все же есть и другой формат хранения изображений – векторный. Вернемся к основной идее сохранения изображения – необходимо сохранять не картинку, а ее элементы. В отличие от растровой графики, разбивающей изображение сеткой на пиксели, в векторной графике происходит разбиение на элементарные геометрические объекты, которые называются примитивами. Это точки, линии, кривые, круги, многоугольники и прочее. Каждый такой примитив состоит из элементарных отрезков кривых, и в памяти компьютера запоминаются их параметры, например тип линии (сплошная или пунктирная), тип заливки, координаты узловых точек, радиус кривизны и так далее.

Теперь, чтобы сохранить изображение треугольника нам достаточно записать координаты начала и конца трех отрезков, его составляющих, их толщину и цвет заливки.

При таком подходе сохранение изображения в памяти компьютера сильно походит на кодирование текстовой информации, каждому качественно новому элементу ставится в соответствие свое кодовое обозначение и дополнительно сохраняются его параметры (координаты и прочие характеристики). Обычно хранение векторных изображений требует много меньше памяти, в отличие от растровых, хотя у этого метода есть недостатки. Во-первых, устройства вывода изображений (мониторы) построены по принципу матрицы пикселей, поэтому для вывода на экран векторная графика предварительно должна быть преобразована в растровую с помощью специальных программ или видеокарт. Во-вторых, векторная графика больше подходит для хранения логотипов, например, в которых четко вырисовываются примитивы, а фотографии с большим количеством объектов и форм и их наложением друг на друга могут потребовать слишком большого числа примитивов для описания, что плохо скажется на объеме памяти. К тому же спецификации векторных форматов гораздо сложнее растровых.

Вообще, преобразование векторного изображения в растровое происходит достаточно просто – разбиением примитивов на пиксели сеткой соответствующих размеров. Но обратный процесс – трассировка – очень ресурсозатратен и сложен. Качество же на выходе может сильно ухудшиться.

Бесспорным преимуществом векторной картинки является отличная масштабируемость, то есть растягивая векторную картинку мы все еще будем видеть четкие линии объектов, в то время как такое же изображение, сохраненное в растровом формате, покроется квадратами.

2.6.Компьютерное представление мультимедийной информации

Под термином мультимедийная информация принято понимать как звуковую, так и видеоинформацию. Хотя последняя представляет собой объединение большого числа изображений и аудио-дорожек.

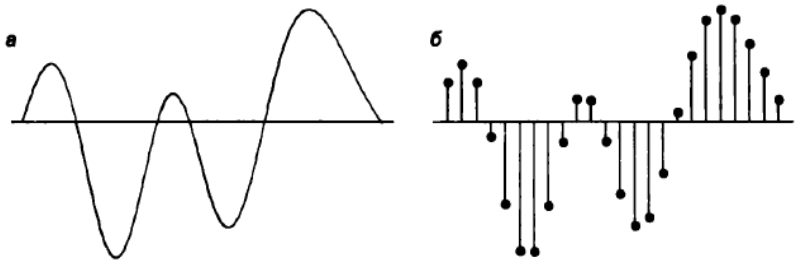
Для начала разберемся с вопросом хранения в памяти компьютера звука. Из физики нам известно, что звук есть упругая волна механических колебаний, у которой непрерывно меняются амплитуда и частота. Увеличение амплитуды мы слышим, как увеличение громкости звука, повышение частоты – как увеличение тона. Хотя на самом деле все гораздо сложнее, так как громкость зависит в том числе и от звукового давления, частоты, а также формы колебаний, а высота звука – не только от частоты, но еще и от звукового давления.

Как и в случае с бесконечным количеством вещественных чисел, которые невозможно сохранить в компьютере, так и с постоянно и непрерывно меняющимися характеристиками звуковой волны – для хранения лишь счетного количества измерений приходится мириться с погрешностью такой обработки и хранения.

В звукозаписывающей аппаратуре существует специальное устройство –АЦП (аналого-цифровой преобразователь) – которое занимается дискретизацией (или оцифровкой). Суть этого

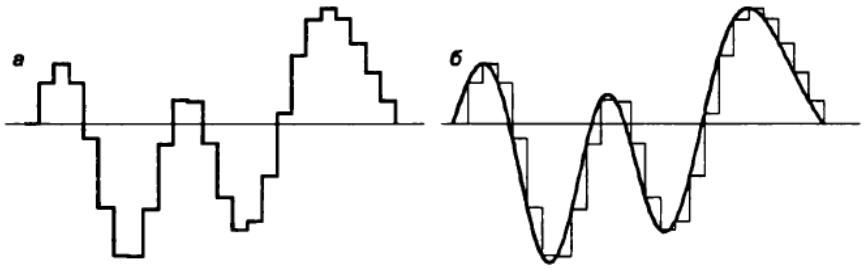
процесса заключается в следующем: непрерывный (или, как его принято называть, аналоговый) звуковой сигнал преобразуется в последовательность электрических импульсов (то есть, двоичных нулей и единиц) за счет кодирования показателями характеристик звука (замеряется период, амплитуда и прочие), которые замеряются через равные промежутки времени. Для хранения этих характеристик их принято усреднять до ближайшего эталонного для хранения показателя и таким образом получаются уже дискретные сигналы (то есть их конечное число, что позволяет их записать в памяти компьютера).

Рисунок 10. Преобразование аналогового звукового сигнала в дискретный:
а – звуковой сигнал на входе АЦП;
б – дискретный сигнал на выходе АЦП



Для воспроизведения (обратного процесса восстановления) звука в аппаратуре служит ЦАП (цифро-аналоговый преобразователь), которых получает ступенчатый сигнал и сглаживает его перед проигрыванием.

Рисунок 11. Преобразование дискретного сигнала в звуковой сигнал:
а – дискретный сигнал на входе ЦАП;
б – звуковой сигнал на выходе ЦАП.



Напомним, некоторые понятия физики.

Период, как характеристика периодического процесса, определяется самым маленьким промежутком времени, за который система совершает одно полное колебание (то есть возвращается в изначальное состояние), то есть период звуковой волны – наименьший промежуток времени между повторениями колебаний звуковой волны в некоторой точке пространства. В Международной системе единиц (СИ) единицей измерения времени является секунда (с), а так как период есть промежуток времени, то и он наследует эту единицу.

Амплитуда, как характеристика волнового процесса, определяется максимальным смещением переменной величины от среднего значения за период, а для звуковых волн и аудиосигналов под амплитудой понимают давление воздуха в волне (называемое эффективным давлением). Для измерения звуковой амплитуды принято пользоваться единицей измерения – децибел (дБ, dB) – это десятая доля логарифма эффективного давления воздуха, хотя в Международную систему единиц эта единица не входит.

Частота, как характеристика периодического процесса, определяется количеством повторений в единицу времени, то есть частота звуковой волны – число колебаний в секунду. В Международной

системе единиц единицей измерения частоты является герц (Гц, Hz). Человеческое ухо воспринимает акустические волны с частотами от 20 Гц до 20 кГц. Этот диапазон называют звуковым.

В ходе оцифровки происходит последовательно три операции: дискретизация по времени, квантование по амплитуде и непосредственно кодирование. Для процессов дискретизации существуют свои важные характеристики, такие как частота дискретизации и глубина кодирования звука.

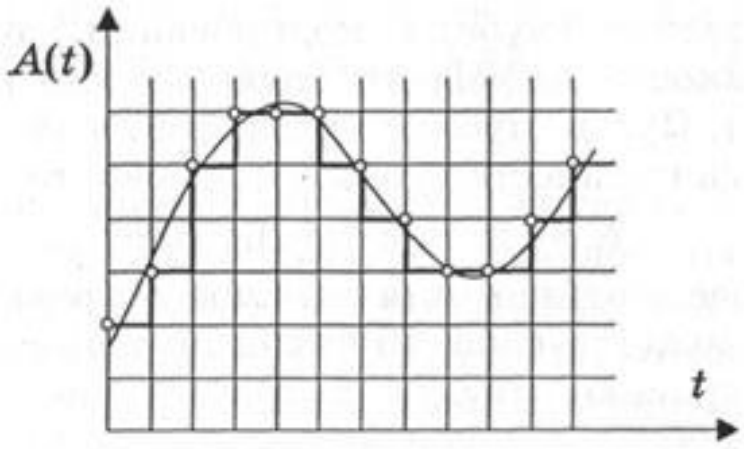
Частота дискретизации – это количество измерений уровней громкости звукового сигнала за одну секунду, причем одно измерение в секунду соответствует частоте 1 Гц. Для более качественной записи нужно производить большее количество измерений за секунду. Стандартные форматы звукозаписи предусматривают частоты дискретизации от 8 кГц, что соответствует частоте радиотрансляции, до 48 кГц, соответствующей качеству звучания современных музыкальных носителей.

Глубина кодирования звука – это количество информации, предусмотренное для кодирования дискретных уровней громкости оцифрованного звука. Наиболее распространены форматы звукозаписи с 16-битной, 32-битной и 64-битной глубиной кодирования. Тут легко провести аналогию с глубиной цветовой палитры, о которой шла речь ранее. Допустим уровни громкости звука – это набор возможных состояний N, для их кодирования необходимо определенное количество информации k. Нетрудно предложить подход, основанный на формуле Хартли: k – минимальная степень двойки, покрывающая набор возможных состояний N (то есть, $2^{k-1} < N \leq 2^k$). Теперь по ходу кодирования каждый уровень громкости звука будет записан как k-битовый двоичный код, наименьший уровень звука получит в соответствие код 000…00 (k штук нулей), а наибольший – 111…11 (k штук единиц).

На деле происходит следующее:

1) Дискретизация по времени – непрерывная звуковая волна разбивается на отдельные маленькие временные участки (вертикальные линии сетки, показатель их количества – частота дискретизации),
2) Квантование по амплитуде – для каждого такого участка устанавливается определенная величина интенсивности звука (горизонтальные линии сетки, показатель их количества – глубина кодирования);
3) Кодирование – в результате, как и в случае с кодированием гладкого треугольника клеточками, гладкая кривая заменяется на последовательность ступенек, двоичное представление которых сохраняется в памяти.

Рисунок 12. Дискретизация звуковой волны



Также на звукозапись влияет количество дорожек или каналов записи. Если записывается только одна дорожка, звукозапись называется монофонической, если больше – стереофонической. Существует и отдельный термин – квадрафоническая запись – для указания того, что запись велась на четыре независимых канала. При подобной записи независимо друг от друга происходит именно

дискретизация для разных каналов, но все показатели остаются одинаковыми, соответственно увеличение количества аудиодорожек записи приводит к кратному увеличению размера файла в памяти носителя.

Зачем же это нужно? Задумывались ли вы, зачем нам два уха? Из-за наличия именно парного слухового органа человек может не просто слышать звук, но и устанавливать направление, в котором находится источник, за счет определения расстояния и угла в горизонтальной и вертикальной плоскостях – это называется бинуральным (или двуушным) эффектом. Стерео-режим позволяет создать для нас звуковую панораму, обманывая наши уши и предлагая нам направления на разные источники, за счет воспроизведения звука по нескольким независимым каналам. Квадро-режим в свою очередь окружает слушателя с четырех сторон и дарит иллюзию присутствия внутри звука.

Что же касается качества звукозаписи, не него влияют как частота дискретизации, так и глубина кодирования. Ниже приведены некоторые современные стандарты:

Низкое качество: запись одной звуковой дорожки (моно-режим), частота дискретизации 8 000 раз в секунду, глубина дискретизации 8 битов (то есть различаются $2^8 = 256$ уровней громкости) – принято в телефонии, так как таких показателей достаточно для хорошей разборчивости речи;

Среднее качество: запись одной звуковой дорожки (моно-режим), частота дискретизации 22 050 раз в секунду, глубина дискретизации 8 битов – принято в радио-режиме;

Высокое качество: запись двух звуковых дорожек (стерео-режим), частота дискретизации от 44 100 до 48 000 раз в секунду, глубина дискретизации 16 битов (то есть различаются $2^{16} = 65\,536$ уровней громкости) – принято в аудио-CD записи на компакт-дисках. Существуют форматы и с удвоенной частотой дискретизации от 88 200 до 96 000 раз в секунду для самой качественной записи звука. Для телевизионного видеосигнала принята частота дискретизации свыше 10 МГц. Обратите внимание, что приставки кило- в словах килогерц и килобайт отвечают за разные кратности! В информатике хорошо известно, что 1 Кбайт = 1 024 байт, однако частота как физическая величина подчиняется классической кратности 1 КГц = 1 000 Гц.

Безусловно в погоне за качеством мы теряем в другом не менее важном показателе – затраты памяти информационного носителя. Информационный объем звукового файла определяется по формуле:

$$V = t \cdot f \cdot k \cdot n,$$

где t – длительность аудиофайла (секунд), f – частота дискретизации (Гц = 1/с), k – глубина кодирования (бит), n – количество записанных дорожек (штук).

Рассмотрим задачу на определение информационного объема: квадрозвук записывался в течение трех минут с частотой дискретизации 32 кГц и 32-битным разрешением без сжатия данных. Определить размер полученного файла в Мбайтах (округлить до сотых).

Частота дискретизации равна 32 кГц, то есть за одну секунду запоминается $f = 32\,000$ значений громкости аудио-сигнала. Глубина кодирования $k = 32$ бита, длительность аудиофайла $t = 3$ мин. $= 180$ с. Количество дорожек $n = 4$. Значит,

$$V = 180 \text{ с} \cdot 32\,000 \frac{1}{\text{с}} \cdot 32 \text{ бита} \cdot 4 = 737\,280\,000 \text{ бит} = 92\,160\,000 \text{ байта}$$

Так как ответ следует дать в Мбайтах, полученный результат следует разделить на 2^{20}. В результате получаем, что в памяти носителя будет записано 87,89 Мбайт.

Рассмотрим подробнее вопрос самого процесса кодирования. Вообще говоря, существуют разные методы для реализации задачи дискретизации. Основные направления: частотно-модуляционный метод (FM-синтез) и таблично-волновой метод (Wave-Table).

В основе FM метода лежит частотная модуляция колебания простой синусоидальной формы. Простыми словами идея заключается в следующем: любой, сколь угодно сложный звук раскладывается на последовательность простейших гармонических сигналов разных частот. Каждый гармонический сигнал представляет собой правильную синусоиду, а значит может быть описан математической функцией. Этим и занимаются АЦП. В результате, изменяя различные параметры гармонических колебаний, можно добиться тембра в очень широком диапазоне от звука

большого колокола до ударных инструментов. Несмотря на нестабильность высоты звука в этом подходе, такое значительное преимущество, как стабильная четкая оцифровка, сделало этот метод стандартом звукозаписи. Он широко использовался в синтезаторах и компьютерных играх. Это метод до сих пор совершенствуется и существует множество вариаций на основе этого принципа.

Метод Wave-Table основан на периодическом воспроизведении произвольной волны с модуляцией формы, высоту тона, продолжительности и интенсивности звука и прочих параметров. Сначала выбирается произвольное положение в волновой таблице – некая форма волны одного цикла. Затем с помощью метода цифровой интерполяции между соседними формами волны происходит динамическое, плавное изменение тембра производимого тона и прочих характеристик. Интерполяция – это процесс поиска промежуточных значений некоторой функции по известному конечному набору ее точек, то есть имея некоторые показатели характеристик в волновой таблице, аппаратура математически рассчитывает и моделирует значения в промежутках между ними при необходимости.

Коснёмся также вопросов обработки звуковых файлов и создания различных звуковых эффектов, таких как, например, фильтрация, очистка звука от нежелательных шумов, изменение тембра и так далее. Наиболее популярные преобразования звуковой информации производятся для изменения следующих характеристик звучания:

Амплитуда: существует необходимость изменять амплитуду по какому-либо закону на определенных участках сигнала (усилять его или, наоборот, ослаблять) для ослабления перекрестных помех;

Частота: сигнал, как набор частот, зафиксированных через определенные промежутки времени, чаще всего подвергается фильтрации для очищения звука;

Фаза: фазовый сдвиг применяется в основном в стерео-записях для создания эффекта поворота, что придает иллюзию «объемного» звука;

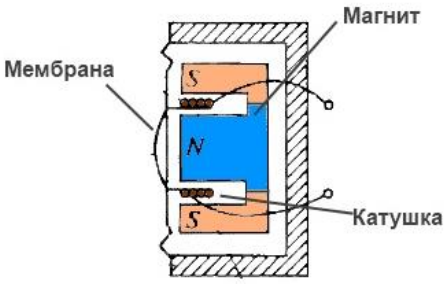
Время: эффекты эха или хора, а также реверберация (ощущение пространственных характеристик звука) достигается за счет наложения, растягивания или сжатия различных звуковых сигналов.

Обсудим также наиболее популярные форматы хранения звуковой информации.

Формат MIDI (Musical Instrument Digital Interface) впервые был разработан для управления и обмена данными между электронными музыкальными инструментами. С весьма точной привязкой ко времени в нем кодируется нажатие клавиш и настройка стандартных акустических параметров (таких как громкость, темп, высота тембра и других). MIDI-команды записываются в порядке изменений соответствующих элементов звука, такая последовательность может быть сохранена на любом цифровом носителе и воспроизведена с помощью секвенсора – специального устройства, по принципу действия похожего на цифровой магнитофон.

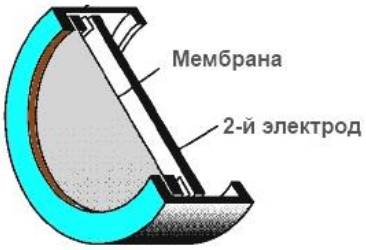
Другой хорошо известный формат записи оцифрованного аудиопотока– WAV (Waveform Audio File Format). Обычно в таком формате хранится цифровое представление исходной звуковой волны без преобразований, в частности сжатия. На сам алгоритм кодирования никаких ограничений не накладывается, по существу, этот формат является лишь контейнером, то есть его спецификации касаются только способа представления данных (размера, структуры, содержания заголовков). Этот формат построен на основе другого формата файлов-контейнеров – RIFF (Resource Interchange File Format), который универсально подходит для любых медиафайлов, в том числе видео (например, на RIFF построен хорошо известный формат AVI) или даже текста. WAV-файлы хранят большое количество дополнительный параметров, таких, как например, метки особенных позиция для синхронизации различных частей звукового файла.

Пожалуй, самый популярный, распространенный и известный среди обывателей формат аудиофайла – MP3 (MPEG-1 Audio Layer 3). Данный стандарт обеспечивает довольно высокое качество кодирование, однако использует сильное сжатие с потерями (примерно в 10-12 раз), что положительно сказывается на информационном объеме занимаемой памяти и отрицательно – на качестве звука. Большие дискуссии о применимости данного формата ведутся в музыкальной

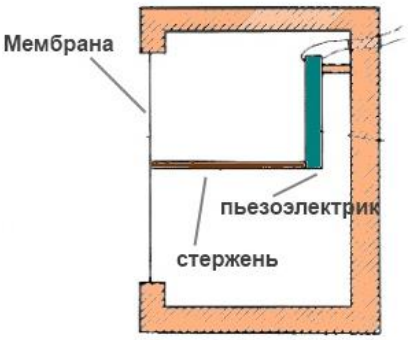


индустрии, хотя для слушателей неискушенных потери в качестве на самом деле не так уж сильно ощутимы.

Теперь поговорим о записи видеофайлов. Тут необходимо неким подобным способом представить оптическую информацию сначала в виде электрических сигналов, а затем закодировать ее двоичным кодом. Преобразование оптического потока в электрические сигналы происходит с помощью видеокамеры, упрощенно ее можно представить в виде набора из четырех устройств:



– Объектив: на предмет, расположенный перед камерой, падает свет, который отражаясь проходит через объектив и проецируя действительное или мнимое изображение (в зависимости от устройства) на матрице – светочувствительном сенсоре;



– Матрица: состоит из большого количества светочувствительных элементов. Именно эти элементы улавливают свет и передают информацию о нем уже в электронном виде в процессор, таким образом формируя видеосигнал или цифровой видеопоток. Далее он конвертируется в аналоговый формат как изображение и усиливается для сохранения. В результате на карте памяти хранятся данные о цвете и яркости отдельных участков изображения в виде кодовых комбинаций электрических импульсов;

– Устройство для получения звукового сигнала (микрофон): вообще говоря, существует несколько самих микрофонов, различных в организации по принципу действия. Однако все они не представимы без мембраны – тонкой гибкой пластинки, закрепленной по периметру. Звук, как колебание воздуха, попадая на мембрану заставляют ее синхронно колебаться, что возбуждает электрические колебания за счет того явления, на котором основан принцип действия конкретного микрофона. Это может быть явление электромагнитной индукции в электромагнитных микрофонах, которое происходит за счет взаимодействия электрического магнита и ферромагнитной мембраны. Или микрофон может быть устроен на принципе изменения ёмкости конденсаторов. В таких микрофонах установлены два плоских электрода, один из которых играет роль самой мембраны. Так двигаясь под действием звуковой волны, подвижный электрод изменяет емкость всей системы. Еще один распространённый вариант – пьезоэлектрический микрофон, в котором мембрана, под действием колебаний звуковой волны, деформирует пьезоэлектрик, от чего на краях пьезоэлектрика возникает напряжение, по форме повторяющее форму самой волны. На картинках в левой колонке можно увидеть примерные схемы устройства электродинамического, конденсаторного и пьезоэлектрического микрофона соответственно;

– Устройство для непосредственного сохранения всех видео- и звуковых данных (карта памяти).

Сам видеофайл представляет собой набор кадров (изображений), которые будут сменять друг друга с установленной частотой при воспроизведении и аудио-дорожек, о которых в отдельности говорилось выше. Однако хранятся эти составляющие не в исходном виде, а в уже сжатом и обработанном, в противном случае, информационный объем таких файлов был бы огромен. Представьте себе, сколько потребуется памяти для хранения хотя бы трех секунд видео в необработанном виде. Стандартно, один кадр представляет собой набор пикселей 720×576, а под хранение одной точки нужно выделить по 8 бит для каждого цвета выбранной модели (допустим, используется модель RGB, тогда необходимо 24 бита). Получается, на хранение одного

необработанного кадра нужно по крайней мере $720 \cdot 576 \cdot 24 = 9\,953\,280$ бит $\approx 1{,}19$ Мбайт. Помимо этого, в одной секунде принято сменять 24 кадра, то есть для нашего гипотетического файла нужно выделить память под минимум 72 изображения, да еще под сопровождающий звук! Как же помещается весь этот объем на современных носителях? Все благодаря разнообразным алгоритмам сжатия информации, которые позволяют экономить память практически без потери качества картинки и звука. Сжатие было бы невозможно, если бы каждый следующий кадр был уникальным, кардинально отличным от предыдущего и расположение пикселей в нем было бы полностью случайным, но это, к счастью, не так – друг за другом идущие кадры могут быть совсем не отличимы или отличимы лишь в контурах двигающихся объектов. Благодаря этому помимо сжатия самого изображения, которое применяется и для самих картинок, можно обрабатывать кадр и убирать из него излишки, которые не будут заметны глазу при быстром воспроизведении, например, чистое голубое небо без солнца и облаков фактически является однородным голубым полем, а значит его можно описать в виде граничных точек и градиента заливки. К тому же можно достаточно сильно сжимать похожие соседние кадры (например, с помощью технологии компенсации движения).

Ранее был уже упомянут файл-контейнер AVI (Audio Video Interleave). Как и прочие контейнеры, этот формат не накладывает ограничений на содержимое, а значит может содержать видео- и аудио-данные, сжатые с использованием разных технологий, не влияющих на синхронность воспроизведения. Что касается технологий видеозаписи, приведем в пример популярную DivX, которая при небольшой потере качества позволяет добиться сжатия в 8-12 раз.

2.7.Количество информации

С самим термином «информация» мы познакомились в предыдущей главе. Теперь перейдем к вопросу, как же ее измерить. В случае с изображением или звуковым файлом задача ясна – разбить на составляющие, округлить показатели и посчитать количество ячеек памяти, которые все эти данные уместят. Но как посчитать количество информации, например в предложении или каком-либо событии. Этот расчет основан на вероятностном подходе, то есть из соображений, что все события имеют определенную вероятность случиться или не случиться. Например, мы подкинули монету, и она упала решкой вверх. Сколько информации мы получили в этом сообщении?

Строго говоря, не в каждом сообщении есть новая информация, в таком случае мы ее не получаем вовсе. Например, в сообщении «мы подкинули монету, и она упала одной из сторон вверх» новой информации не содержится, потому что этот исход очевиден, и мы в нем, итак, не сомневались (будем считать, что на ребро она встать никак не могла). Однако, если, узнав некий факт, вы уменьшили неопределенность ваших знаний, то, очевидно, новая информация получена и ее можно измерить. Изначально неизвестно, как именно упадет монета, после известия мы уменьшаем неопределенность – узнаем точную сторону, на которую упала монета, значит информация получена.

В конечном счете было всего два варианта «решка» или «орел», после полученной новости мы знаем точный один вариант «решка». Значит полученная информация уменьшила неопределенность ровно в два раза. Такое сообщение оценивается в 1 бит – минимально возможное количество информации. Бит – единица измерения количества информации и определяется именно как количество информации, которое содержится в информационном сообщении, уменьшающем неопределенность знания в два раза

Очевидно, что количество информации напрямую зависит от количества возможных исходов ситуации (в случае равновероятных событий), чем больше неопределенности вначале, тем больше информации будет получено в конце. Если есть стопроцентный факт, например солнце встает на востоке (вероятность этого события – 100%), то ежедневный восход не дарит нам новых знаний и количество информации такого события равно 0. Между прочим, тот факт, что солнце никогда не встанет на западе также неоспорим (вероятность такого события – 0%), а значит сообщение «сегодня солнце опять не встало на западе», также не несет никакой информации. В обоих случаях неизменно одно – мы не сомневались в исходе.

Сам термин «количество информации» в теории информации принято понимать как меру уменьшения неопределенности знания при получении сообщений – такой подход был предложен американским математиком Клодом Элвудом Шенноном, он назвал количество информации – информационной энтропией (по аналогии с энтропией термодинамики).

Начнем знакомство с вычислением этой характеристики с другого американского математика, не раз упомянутого уже в данной главе – Ральфа Винтона Лайона Хартли. Его подход касается равновероятных событий. Пусть существует N возможных событий, каждое из которых может произойти с равной вероятностью $p = \frac{1}{N}$. При наступлении одного конкретного события из них, мы получим сообщение, которое несет в себе количество информации $I = \log_2 N$.

Ранее мы ссылались на эту формулу в виде $N = 2^I$ для определения, например глубины палитры. На самом деле, выбирая из 8, скажем, цветов один конкретный для данного пикселя, мы получим $I = \log_2 8 = 3$ бита информации (именно столько понадобится для хранения выбранного нам цвета в памяти компьютера). Обратите внимание при этом, что любой из восьми цветов мог быть выбран с равной вероятностью (что важно для подхода Хартли).

В случае, если количество событий не равно в точности степени двойки, количество информации принято округлять в большую сторону. Вообще говоря, в теории информации допустимо указывать количество информации и как не целое число для большей точности, однако в вычислительной технике это невозможно (как можно занять одну десятую бита), поэтому и прибегают к округлению. Например, если цветов в палитре 10, то необходимое количество информации

$$I = \lceil \log_2 10 \rceil = \lceil 3{,}32 \rceil = 4 \text{ бита}.$$

Теперь приступим к более общему подходу – расчет количества информации по формуле Шеннона. Пусть существует N возможных событий, каждое из которых может произойти с вероятностью соответственно $p_1, p_2, \dots, p_{N-1}, p_N$. При наступлении одного конкретного события из них, мы получим сообщение, которое несет в себе количество информации

$$I = p_1 \cdot \log_2 \frac{1}{p_1} + p_2 \cdot \log_2 \frac{1}{p_2} + \cdots + p_{N-1} \cdot \log_2 \frac{1}{p_{N-1}} + p_N \cdot \log_2 \frac{1}{p_N} == \sum_{1 \le i \le N} p_i \cdot \log_2 \frac{1}{p_i}$$
$$= -\sum_{1 \le i \le N} p_i \cdot \log_2 p_i$$

Знак минус появляется в последнем преобразовании за счет свойства логарифма, так как $\log \frac{1}{a} = \log a^{-1} = -\log a$. Рассмотрим ситуацию с равновероятными событиями, согласуются ли эти два подхода. В данном случае вероятность каждого события равна: $p_1 = p_2 = \cdots = p_i \dots = p_N = \frac{1}{N}$. Тогда

$$I = \frac{1}{N} \cdot \log_2 \frac{1}{\frac{1}{N}} + \frac{1}{N} \cdot \log_2 \frac{1}{\frac{1}{N}} + \cdots + \frac{1}{N} \cdot \log_2 \frac{1}{\frac{1}{N}} = \log_2 N \underbrace{\left(\frac{1}{N} + \frac{1}{N} + \cdots + \frac{1}{N}\right)}_{N \text{ штук}} == \log_2 N \cdot \frac{1}{N} \cdot N$$
$$= \log_2 N$$

Значит эти два подхода согласованы.

Существует еще и алфавитный подход измерения количества информации сообщений. Этот метод предложен советским ученым – Андреем Николаевичем Комогоровым. В этом случае принято абстрагироваться от смысла сообщения и рассчитывать чисто технический параметр, который необходим в теории информации. Раз мы теперь не обращаем внимание на содержание, мерилом для нас будут выступать элементы этого сообщения – символы. Не стоит ошибочно

Символ	Частота, %
Пробел	14,46
О	9,42
Е	7,33
И	6,72
А	6,52
Н	5,83
Т	5,56
С	4,69
Р	4,05
В	3,89
Л	3,77
К	2,98
М	2,74
Д	2,54
П	2,39
У	2,23
Я	1,70
Ы	1,61
Ь	1,47
Г	1,43
З	1,39
Б	1,34
Ч	1,21
Й	1,00
Х	0,79
Ж	0,77
Ш	0,58
Ю	0,51
Ц	0,37
Щ	0,26
Э	0,23
Ф	0,18
Ъ	0,03
Ё	0,01

воспринимать за значимые элементы только буквы алфавита, очень важное значение играет и пробел – самый частый символ любого языка, и прочие знаки (цифры, знаки препинания). Для расчёта количества информации важно все.

В левой колонке предлагается частотное распределение символов русского языка (33 буквы и пробел). Частота – это усреднённое количество встреч данного символа в процентах среди среднестатистического сообщения. Для получения вероятности следует разделить частоту на 100.

В теории информации такой набор символов принято называть алфавитом сообщения (не путать с алфавитом языка). Количество символов в нем (N) называют мощностью алфавита. Обратите внимание, в конкретном сообщении могли участвовать не все буквы алфавита языка и тем не менее при расчете мы их тоже должны учесть. Таким образом в алфавит сообщения входят все символы, которые могли быть использованы в подобного рода сообщениях и которые вы решите включать в расчеты.

Далее после определения мощности алфавита вашего сообщения следует применить один из подходов (Хартли или Шеннона) для определения количества информации, приходящегося на один символ сообщения. Если все символы равновероятны (что в естественных языках не встречается, но в учебных задачах допустимо), то используется подход Хартли и на один символ приходится

$$K = \log_2 N \text{ бит нформации.}$$

Если же мы знаем частотное распределение алфавита, то используем подход Шеннона

$$K = \sum_{1 \le i \le N} p_i \cdot \log_2 \frac{1}{p_i}$$

Частотное распределение представляет собой таблицу из двух строк – в одной указаны все символы алфавита сообщения, в другой – вероятность, с которой эти символы могут оказаться в среднестатистическом сообщении.

В конце концов, чтобы определить количество информации во всем сообщении, нужно умножить количество символов в нем n на количество информации одного символа:

$$I = K \cdot n.$$

Частотное распределение символов считается на основе анализа большого количества текстов данного языка. Самый известный ресурс для русского языка — это Национальный корпус русского языка (НКРЯ), на сайте которого (http://www.ruscorpora.ru) можно найти много статистической информации о строении нашего языка.

Напоследок заметим, что при расчете количества информации о наступлении нескольких независимых событий следует рассчитать количество информации для каждого в отдельности и результаты сложить.

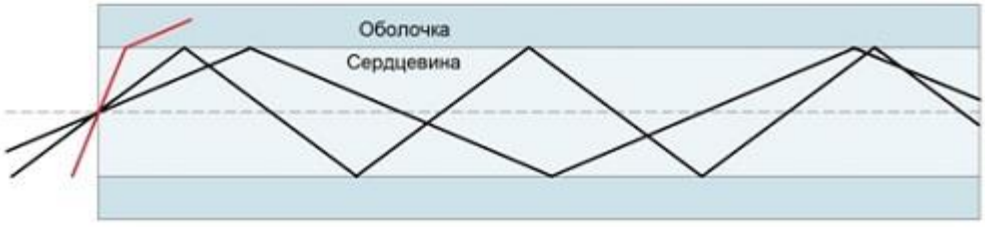
Раздел 3. Передача информации

3.1. Канал связи

В предыдущем разделе мы познакомились с вопросами хранения данных на цифровом носителе. Теперь рассмотрим вопрос распространения и передачи этих данных. В технике принят термин – канал связи. Он служит для обозначения сложно сконструированной системы для передачи данных на расстояние от отправителя, который называют источником, к получателю, которого называют приемником. Эта система состоит из необходимых технических средств и непосредственно среды распространения сигнала. В зависимости от типа этой среды, от ее физической природы различают разные каналы связи:

- Акустический канал: мы используем его регулярно при устном общении друг с другом, так как информация из нашей головы попадает в голову собеседника именно в виде звуковой волны, распространяемой в воздухе. Более формально принято вести речь об акустическом волноводе. Это участок некой однородной среды, способной распространять звук, ограниченный стенками или другими средами, чтобы ограничить расхождение волн в разные стороны. В качестве примера можно привести водоем это большой естественный волновод, ограниченный снизу грунтом и сверху – поверхностью воды, что позволяет звукам низких частот распространяться в воде. Примером искусственного акустического волновода может стать музыкальный духовой инструмент, например, дудук. Из определенной древесины изготавливается трубка, воздух в которой распространяется в одном направлении за счет многократного отражения от стенок.
- Оптический канал: этот канал мы используем еще чаще – каждый раз, когда смотрим на что-то. Лучи света, отраженные от поверхности предмета, и попадают нам в глаз, преобразуются летками сетчатки в нервные импульсы. Эти импульсы отправляются в мозг и формируют зрительный образ. В технике принято различать наряду с беспроводным оптическим каналом также и проводной оптический канал. В беспроводном виде связи распространение сигнала происходит через атмосферу за счет лазерного луча (который испускает электромагнитные волны оптического диапазона в направлении приемника – высокочувствительного фотодиода). Такой принцип замечательно подходит для связи в космосе. Сигнал лазерного излучателя можно принимать приемником даже на расстоянии более 20 миллионов километров. Проводной канал представляет собой особую конструкцию – оптическое волокно. Это нить из прозрачного материала, например кварцевого стекла, которая может переносить свет внутри себя за счет полного внутреннего отражения. Как звуковая волна отражается от стенок трубки музыкального инструмента, так и электромагнитная волна отражается от стенок световода. Как правило, оптическое волокно состоит из двух компонент: сердцевину делают из чистового материала, а оболочку – из материала с добавками для изменения показателя преломления (показатель преломления оптической оболочки всегда должен быть немного меньше, чем показатель преломления сердцевины, примерно на 1 %). Таким образом, луч света, направленный в сердцевину, сможет распространяться в ней, постоянно отражаясь от оболочки.

Рисунок 13. Принцип действия оптического волокна



- Проводной канал: данный вид канала связи принято называть кабелем. Кабель – это

направленная среда передачи электромагнитного сигнала. Кабель состоит из изолированных проводников, которые принято называть жилами, заключенных в оболочку, которую называют изоляцией. Также в конструкции могут присутствовать экран (защита от внешних электромагнитных полей), броня (защита от механических деформаций) и гидрофобный заполнитель (защита от влаги). Несколько изолированных жил объединяются в сердечник. Для передачи электрической энергии в качестве материалов для жил используют медь, алюминий, сталь и прочие металлы. Для передачи оптических сигналов – стекло и различные пластмассы. Оболочка может быть из ткани, пластмассы, металла, резины и прочее.

Рисунок 14. Конструкция оптоволоконного кабеля

Связь же осуществляется не просто за счет отправления по проводу сигналов. Необходимо наладить сложную систему, включающую в себя не только кабель, но и узлы связи и регенерационные пункты. Узлы связи необходимы для переноса данных из одного кабеля в другой, а регенерационные пункты – для усиления сигнала, который неизбежно затухает при распространении на дальние расстояния. Хорошо известным примером использования такого вида связи является телефония. Мы произносим в трубку некоторые слова, которые преобразуются в электрический сигнал и передаются через телефонную сеть другой стороне – нашему собеседнику. На том конце происходит обратный процесс преобразования электрического сигнала в голосовой. Телефонная сеть представляет собой разветвленную систему автоматических телефонных станций (устройств коммутации и соединения абонентов), связанных между собой медными или оптоволоконными кабелями и спутниками связи.

- Инфракрасный канал: этот канал связи не требует использования проводов и не чувствителен к электромагнитным помехам. Принцип работы основан на передаче данных за счет невидимого инфракрасного излучения.
- Радиоканал: этот канал также является беспроводным, однако его недостатком выступает высокая чувствительность к помехам. Организован он на коротких и ультракоротких волнах, по высоким и сверхвысоким частотам.

Принято различать две формы передачи сигнала: аналоговую (то есть сигнал непрерывен и представляет собой некую функцию по времени) и дискретную (цифровой сигнал в импульсной форме, воспринимаемый как набор отдельных значений).

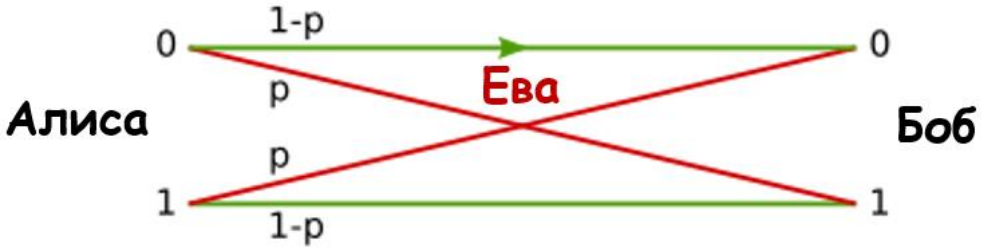
Упрощенная схема системы связи включает в себя как минимум три составляющие: источник данных, канал передачи информации и приемник. Это весьма примитивный взгляд, на практике, между источником и каналом и между каналом и приемником располагается несколько специализированных, обычно парных устройств обработки сигнала. Об этих устройствах речь пойдет позже.

Рисунок 15. Примитивная схема канала связи

Напоследок упомянем про еще один примитив, часто используемый теоретиками в изучении вопросов кодирования информации. Двоичный симметричный канал связи (ДСК) – упрощенная модель канала связи, который принимает двоичный код (последовательность из 0 и 1) и с некоторой вероятностью неизменными возвращает их приемнику.

Выбранная вероятность искажения – это модель эффекта наложения помех, которые всегда присутствуют при передаче. В технике для них принят термин – шум. Это хаотические колебания определенной физической природы (зависящей от среды распространения в используемом канале связи), оказывающие нежелательное воздействие на передаваемую информационную последовательность. Между прочим, в теории защиты информации этот шум также играет роль Евы (то есть злоумышленника), которая нарушает одно из важнейших свойств информации – целостность. Пусть вас не смущает этот факт, злоумышленник не обязан быть человеком, это может быть и природа.

Рисунок 16. Схема двоичного канала связи



Выше представлена схема такого ДСК. Такие каналы принято описывать вероятностью передачи искаженного сигнала, которую называют вероятностью ошибки одного символа. Очевидно, возможны только два исхода событий: успешно передано или передано с искажением. Это полная система, то есть сумма вероятностей этих исходов должна составить единицу. Отсюда следует, что если ошибочный сигнал придет с вероятностью *p*, то успешная передача сигнала происходит с вероятностью *1-p*. Проще говоря, при отправлении единицы приемник получит единицу с вероятностью *1-p* (зеленая линия) и ноль с вероятностью *p* (красная линия). Аналогичная ситуация с отправкой по такому каналу связи нуля.

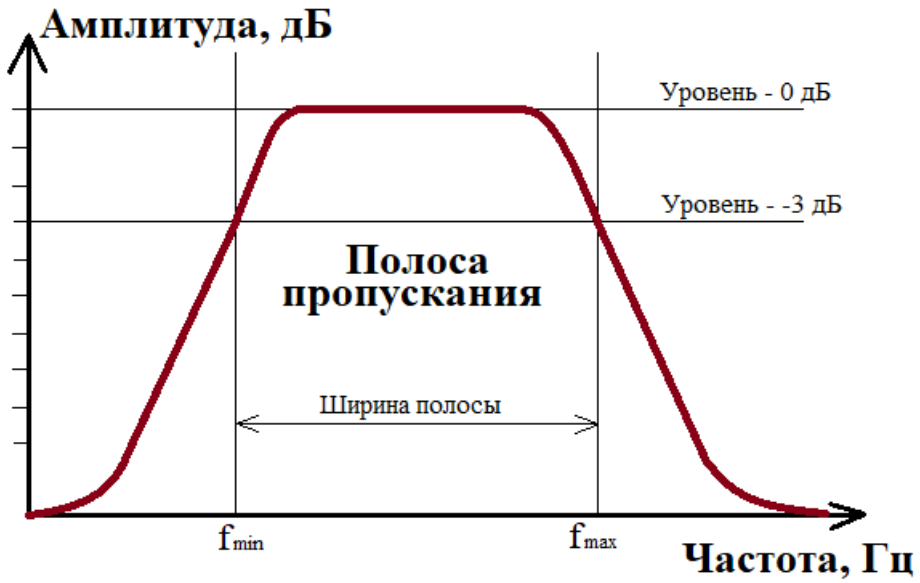
3.2.Характеристики канала связи

Оценивать и сравнивать между собой каналы связи можно по многим параметрам. Одни отвечают за показатели из области физики, например помехозащищенность и полоса частот, другие – из области теории информации и кодирования, в частности важнейшими являются пропускная

способность и скорость передачи информации. Рассмотрим некоторые из таких параметров.

Одним из физических показателей является ширина полосы пропускания, которая используется в теории антенн. Сама полоса пропускания – это непрерывный диапазон передаваемых каналом частот без существенного искажения информационного сигнала. Обычно отношение амплитуд выходного сигнала ко входному в этой полосе не должно превышать 0,5. Для любого канала, будь то радиотехнических, акустический или оптический, можно построить амплитудно-частотную характеристику (АЧХ). Это график, на горизонтальной оси которого стоит частота, а на вертикальной – амплитуда. По нему оценивается полоса, в которой линия графика не претерпевает сильного скачка, показывает достаточно равномерное поведение. Вычислять ширину этой полосы принято так: на графике амплитудно-частотной характеристики проводится линия на уровне -3 дБ (это допустимый уровень неравномерности при передаче сигнала, нулевым уровнем является уровень самой большой амплитуды) и определяется две границы частот (нижняя и верхняя) для этой линии. Их разность называют шириной полосы пропускания. Эта характеристика соответственно также измеряется в единицах частоты, то есть в Гц. В телефонной связи эффективно передаваемая полоса частот – это частоты от 0,3 до 3,4 кГц, в телевизионном вещании – от 12,2 до 12 388 кГц.

Рисунок 17. Амплитудно-частотная характеристика



Также по амплитудно-частотной характеристике принято считывать такой показатель, как затухание – это одна точка на линии АЧХ. Данный показатель отражает, как сильно уменьшится амплитуда (или мощность) сигнала во время передачи по каналу определенной частоты. Таким образом можно предварительно оценить уровень искажения сигналов. Как правило затухание измеряется также в дБ и рассчитывается так:

$$A = 10 \cdot \lg \frac{P_{вых}}{P_{вх}},$$

где P – мощность сигнала (соответственно выходного и входного). Эта величина всегда отрицательна, потому что в любом кабеле будет происходить затухание и на выходе мощность будет меньше. Для исправления ситуации используют специальные устройства – усилители.

Другой важной технической характеристикой передачи информации является помехоустойчивость линии связи. Простыми словами под помехозащищенностью принято понимать способность линии противостоять помехам внешней и внутренней среды. Безусловно, это зависит и от самой физической природы, на которой основан данный канал связи, и от применяемой конструкции кабелей и прочих каналов, используемых для передачи (имеется в виду аспекты экранирования, брони и прочего подавления помех в устройстве самого канала). Например, для

уменьшения помех от внешних электромагнитных полей проводники принято скручивать между собой. В результате получается так называемая витая пара.

Об этой стороне вопроса можно говорить с точки зрения разных показателей. Мы рассмотрим только два – NEXT и BER.

Термин NEXT – это аббревиатура от Near End CrossTalk, что переводится как перекрёстные наводки на ближнем конце. Он отражает помехоустойчивость витой пары именно к внутренним источникам помех, поскольку электромагнитный сигнал передается по паре проводников, каждый из которых обладает своим полем, неизбежно взаимное наведение помех. NEXT трактуется как внутренняя помехозащищённость и определяется минимальным отношением сигнал/шум. Формула для расчета такова

$$\mathrm{NEXT} = 10 \cdot \lg \frac{\mathrm{P_{вых}}}{\mathrm{P_{нав}}},$$

где P – все также мощность сигнала (соответственно выходного и наведенного). Очевидно, лучшая витая пара обладает меньшим показателем NEXT. Для кабелей прочего строения этот показатель не применяют, так как внутренние наводки в них практически отсутствуют вследствие физической природы (в случае оптоволоконных кабелей) или сильного экранирования.

Термин BER – это также аббревиатура от Bit Error Rate, что переводится как частота ошибок на бит. В русской терминологии этот показатель принято называть достоверностью передачи или интенсивностью битовых ошибок. Эта величина представляет собой вероятность получения искаженного бита. Искажение может произойти как из-за помех, так и из-за ограничения полосы пропускания сигнала.

Теперь перейдем к характеристикам теории информации. Важнее всего рассмотреть в этом вопросе скорость информационного потока (или скорость передачи данных) – это объем данных, переданных каналом связи за единицу времени (чаще всего секунду). Измеряется эта величина в битах в секунду, что записывается как bps (bits per second) или в байтах в секунду – Bps (bytes per second). Также можно встретить еще одну интересную единицу измерения – бод. Однако бод – это не количество бит, переданное за секунду, а количество символов (что одно и то же только для двоичного кодирования). Подробнее эту величину мы будем рассматривать несколько позже.

Сам показатель скорости передачи данных увеличивать бесконечно нельзя. В теории информации существует теорема Шеннона-Хартли о верхней границе скорости передачи данных, которая описывает зависимость предельно допустимой скорости от полосы пропускания канала, мощности сигнала и показателя шума. Максимальную же скорость передачи данных в канале связи называют пропускной способностью канала.

С одной стороны, чтобы увеличить пропускную способность канала следует увеличить частоту несущего сигнала, однако это повлечет за собой расширение спектра сигнала и увеличение количества искажений, то есть скорость передачи полезной и правильной информации на самом деле не увеличится при таком подходе. Из теоремы Шеннона-Хартли следует, что необходимо либо увеличить мощность источника, либо уменьшить мощность шума. Однако это нелегко, поскольку весьма затратно и не эффективно из-за логарифмической зависимости показателей.

Для телефонии организованы каналы связи с пропускной способностью от 2 400 до 9 600 bps. Такую скорость обеспечивают витый пары. Для высокоскоростных каналов этот показатель значительно выше – от 56 000 bps. Такие каналы конструируются из экранированных оптоволоконных кабелей.

Вспомним также об особенном теоретическом канале – двоичный симметричный канал связи. Его пропускная способность рассчитывается по отдельной формуле с использованием двоичной энтропии. Этот метод учитывает подход Шеннона к расчету количества информации. Как мы говорили ранее ДСК характеризуется вероятностью ошибки одного символа – p. Итак, в нашем алфавите есть два символа 0 и 1. Чтобы не отправил источник данных, с некоторой вероятностью мы можем все равно получить любой из этих символов. Согласно формуле Шеннона, двоичная энтропия ДСК с вероятностью ошибки p определяется так:

$$H_p = -p \cdot \log_2 p - (1-p) \cdot \log_2 (1-p)$$

В свою очередь пропускная способность такого канала определяется как противоположная характеристика $C = 1 - H_p$.

Напомним, что несмотря на вид формулы, энтропия – величина положительная, поскольку вероятность – число в отрезке от 0 до 1, то логарифм вероятности будет отрицательным и минус в формуле как раз поменяет знак.

Финальной характеристикой канала связи, с который мы познакомимся, станет объем переданной информации. Вычисляется этот показатель достаточно логично: если канал передает данные со скоростью q (пропускная способность) в течение времени t, то всего будет передано

$$V = q \cdot t \text{ бит информации.}$$

3.3.Типы связи

В теории информации принято выделять три типа связи: симплекс, дуплекс и полудуплекс.

Если ваш канал позволяет передавать информацию лишь в одном направлении, то это симплексная связь. Примером могут служить каналы, используемые в теле- или радио- вещании. По ним сигнал от вышки может дойти только до приемника, больше никак.

Если же канал позволяет передавать сообщения в оба направления, то говорят о дуплесной связи. Например, телефоны позволяют нам и говорить (то есть передавать информацию), и слушать (то есть принимать ответ). Обычно такие устройства одновременно выполняют прием и передачу по двум разным каналам связи (используя отдельные проводники или полосы частот).

Промежуточная между ними форма – полудуплекс. Зачастую она организуется на одном канале связи, настроенном таким образом, чтобы он мог и передавать, и принимать информацию, но только в одном направлении на данный момент. Таким образом, сначала нужно завершить, например, передачу, чтобы потом начать прием. Хорошо известные всем рации являются именно полудуплексом: каждый говорящий может или говорить, или слушать.

3.4.Вычислительная сеть

В прошлом веке люди изобрели такую технологию, которая позволила объединить вычислительные устройства за счет подключения их друг к другу. Конечно, по мимо физического соединения (в виде провода) нужно еще и большое количество специально разработанных программ, обрабатывающих запросы устройств к друг другу. Такие программы и правила, в них описанные, называют сетевыми протоколами. Эта технология развивается и по сей день. Но сегодня уже трудно представить процесс, который бы не задействовал возможности компьютерных сетей.

Компьютерная сеть – это набор технических средств, соединенных различными средствами связи и программным обеспечением для организации информационных процессов.

В 1969 году в Америке появилась первая (из известных и открыто опубликованных) компьютерная сеть ARPANET. Создавали ее, конечно, для военных нужд. На данный момент технологии продвинулись далеко вперед, и организация ARPANET выглядит наивно, например, в ней не было привычной нам сегодня идеи иерархии, т.е. все компьютеры были связаны друг с другом на равных. Сегодня же практически каждая сеть выделяет специальный компьютер-сервер, отличающийся от прочих компьютеров-клиентов мощностью и функционалом. Такое присуще не только специализированным сетям (коммерческим, военным и проч.), но и сетям массового обслуживания.

Сервер – это специальное вычислительное устройство, выделяемое из общей сети для ее обслуживания (запуска необходимых приложений, обработки информационных запросов, обеспечение связи с прочими устройствами вне сети).

Клиент (рабочая станция) – это вычислительное устройство конечного пользователя, подключенное к сети, которое позволяет пользователю обращаться к серверу с различными

запросами.

Из ключевых особенностей организации иерархии компьютерной сети можно привести в пример локализацию обработки запросов:

Если выбрана стратегия централизованной обработки информации, то большая часть работы ложится именно на сервер, а клиенты выполняют лишь то, что не требует большой затраты ресурсов (память, времени, процессорной нагрузки).

Если же выбрана распределенная обработка (или децентрализованная), то сервер выполняет роль организатора распределения задач по рабочим станциям и хранилища, в котором аккумулируются промежуточные результаты.

Также сети принято классифицировать по степени удаленность станций, так как сетевым технологиям не мешает расстояние и объединять в сеть можно технику и на разных концах планеты. Итак.

Если все рабочие станции расположены на небольшом расстоянии друг от друга, то принято называть такую сеть локальной. Это может быть объединение вашего компьютера с периферийными устройствами (принтером например), или другими компьютерами. Такая организация устройств обычно создается для совместной обработки информационных запросов, использования программных средств, локализации систем безопасности.

Среди всех компьютеров можно выделить особый – сервер, а можно и не выделять, тогда сеть будет одноранговой. В таких сетях каждый клиент может выполнять роль сервера или обращаться к другой станции, как к серверу. Конечно, это негативно сказывается на производительности, потому стабильнее настраивать локальные сети, выделяя сервер.

У крупных организаций, которые имеют филиалы в различных городах, есть потребность связывать локальные сети своих подразделений в единую компьютерную сеть. Таким образом организованы корпоративные сети.

Корпоративная сеть – объединение локальных сетей одной организации для совместной обработки информации и использования ресурсов. Чаще всего их организуют с централизованной обработкой данных.

Самой популярной и используемой сетью является, конечно, глобальная сеть, объединяющая устройства по всему миру. Все сети, которые имеют внешнее подключение, имеют выход в глобальную сети и связаны меж собой через нее. Например, отправляя письмо по электронной почте, мы обращаемся к почтовому серверу по сети.

Для связи корпоративных сетей создана технология – Интранет. Интранет для большей эффективности использует распределённую среду (протоколы и технологию Интернет) и построена на технологии «клиент-сервер» с централизованной обработкой информации.

Другими популярными сетями, например, среди телекоммуникационных, являются малые компьютерные сети, или BBS. Они состоят всего из одной рабочей станции. По сути, это общедоступная библиотека файлов, в которых может храниться что угодно, от программного обеспечения, до сервисных и информационных услуг.

Глобальная сеть – это объединение компьютеров на большом расстоянии, организованная для общего пользования мировыми информационными ресурсами. Под термином Интернет принято понимать единые правила, по которым организовано взаимодействие: способ подключения компьютера или локальной сети к глобальной, правила передачи данных, система идентификации компьютера в сети (сетевой адрес).

Также упомянем о специальной технологии децентрализованной обработки информации в сети. Это мощный инструмент, обеспечивающий надежность связи. Для этого используется большой набор специальных компьютеров – мощных узлов связи, которые соединены между собой каналами связи (выделенными телефонными, волоконно-оптическими или беспроводными) и обеспечивают круглосуточный обмен информацией между пользователями сети. Если один узел выходит из строя, то данные направляются по другому маршруту в обход неисправного узла. Технология, поддерживающая, такую реализацию поиска маршрута и доставки данных, как раз обеспечивает надежную связь пользователей несмотря на временные неполадки оборудования.

3.5.Протоколы передачи данных

Как правило в процессе передачи информации данные преобразуются. Это преобразование происходит в строгом соответствии с принятыми протоколами передачи. Формат данных и способ передачи информации определяются Протоколами передачи и являются совокупностью правил, которые принимаются для уменьшения риска возникновения ошибок при передаче.

Для больших информационных объемов используются специальные международные протоколы, осуществляющие блочную передачу информации, коррекцию ошибок, повторы передачи, восстановление сеанса связи после сбоев и т. п.

Коснемся некоторых важных терминов передачи данных.

Пакет. При передаче данных файл принято делить на части – пакеты и и отправлять их по очереди, таким образом сеть не занимается надолго. Но нужны специальные правила упаковки данных, чтобы получатель мог восстановить файл по его фрагментам. Каждый пакет будет направлен отдельно и в итоге может прийти к получателю своим маршрутом, может даже и потеряться. В таких случаях предусмотрен запрос отправителю о потере пакета для повторной отправки. Для контроля надежности передачи используется транспортный протокол TCP.

Маршрутизация. Это операция определения следующего звена в цепочке для передачи данных конечному получателю. Выполняют ее специальные устройства – маршрутизаторы. Для оптимальной отправки маршрутизатор должен искать в своем списке связанных устройств то, которое ближе к получателю. Однако не так прост этот поиск, потому то самый короткий по количеству промежуточных узлов маршрут может оказаться не самым оптимальным (так как у узлов есть свои характеристики передачи данных, такие как например, пропускная способность, надежность передачи). Правила, по которым происходит маршрутизация, составляют протокол IP.

Принято даже выделять отдельным термином набор протоколов TCP/IP. Хотя, конечно, на деле используется гораздо большее количество протоколов со своими функциями для соединения двух устройств.

Сам термин «протокол», как было сказано ранее, подразумевает некоторые правила, настроенные между программами вычислительных устройств в сети, которые регламентируют единый порядок обмена данными между ними и реагирование на нестандартные ситуации. Существует международная организация International Standardization Organization, которая занимается стандартизацией. Во второй половине прошлого века она подготовила стандарт, который до сих пор используется - Open System Interconnection, или OSI – модель связи открытых систем. Эту модель принято назвать стеком (набором) протоколов, которые описывают правила передачи данных в разнообразных сетях во время сеанса связи. Делить эту модель принято на семь уровней по задачам:

- Прикладной уровень. Тут определяется вид обработки запроса от прикладной программы, обратившейся за соединением с другой рабочей станцией
- Уровень представления данных. Производится изменение представления или формата

данных, чтобы процесс-получатель мог ее воспринять.

- Сеансовый уровень. Тут определяется формат передачи данных, устанавливается процедура проведения сеансов и синхронизируется коммуникация.
- Транспортный уровень. Эти протоколы нужны для непосредственной передачи пакетов по сети. Тут пакеты данных разбиваются на блоки и контролируется отправка и получение каждой части отдельно.
- Сетевой уровень. Тут происходит непосредственное обслуживание каналов, соединяющих рабочие станции сети. Протоколы этого уровня должны выполнять задачу маршрутизации.
- Канальный уровень. Протоколы этой группы должны проверить свободна ли среда распространения сигнала и затем передавать кадры (на которые разбиваются пакеты данных) протоколам следующего уровня, предварительно закодировав их помехоустойчивым алгоритмом.
- Физический уровень. Тут происходит непосредственное физическое кодирование кадров (преобразование в оптические или электрические сигналы) и их передача по среде распространения.

Ниже представим ознакомительную таблицу, в который предложены некоторые функции по уровням.

Уровень модели OSI	Примеры протоколов	Некоторые функции
Прикладной уровень	FTP, SMTP, HTTPs	Идентификация пользователя; Передача файлов; Определение соразмерности ресурсов; Согласование ограничений синтаксиса.
Уровень представления данных	SSL, TLS	Согласование и реализация представления данных между прикладными процессами; Представление графического материала (чертежей, рисунков, схем); Шифрование данных
Сеансовый уровень	PAP, PPTP, L2TP	Установление и завершение соединения между системами; Выполнение обмена данными между прикладными процессами; Синхронизация сеансовых соединений; Сообщение прикладным процессам об ошибках; Прерывание прикладного процесса и его корректное возобновление; Завершение сеанса без потери данных.
Транспортный уровень	TCP, UDP	Управление передачей пакетов по сети; Обеспечение целостности пакетов; Обнаружение ошибок и сообщение о них; Возобновление передачи данных после отказов; Подтверждение отправителю передачи данных.
Сетевой уровень	IPv4, IPv6, ICMP	Создание сетевых соединений; Идентификация портов связываемых устройств; Обнаружение и исправление ошибок, возникающих при передаче через коммуникационную сеть; Организация последовательностей пакетов; Маршрутизация и коммутация.
Канальный	Ethernet,	Организация и передача кадров (блоки, на которые

уровень	PPP	разбиваются пакеты); Обнаружение и исправление ошибок передачи.
Физический уровень	Ethernet	Передача и прием сигналов в последовательном коде; Прослушивание каналов (в случаях, когда это необходимо); Идентификация каналов; Оповещение об ошибках.

3.6.Коммутация и маршрутизация

Коммутация в компьютерной сети – это процесс соединения абонентов такой сети через транзитные узлы. Абонентами могут выступать вычислительные устройства, локальные сети, телефоны и прочее. Обычно двух абонентов не получается соединить персональной физической линией связи специально только для них, поэтому прибегают к коммутации, чтобы разделить физические каналы для использования разными абонентами во время их сеансов связи.

Каждый абонент соединяется с коммутатором индивидуальной линией связи. Линии связи, протянутые между коммутаторами, могут использоваться несколькими абонентами совместно. Коммутация по праву считается одной из самых популярных современных технологий, оставляя другие технологии организации связи в стороне. Популярность коммутаторов обусловлена прежде всего тем, что они позволяют за счет разделения сети на более мелкие подсети повысить ее производительность. Помимо разделения сети на мелкие сегменты, коммутаторы дают возможность создавать логические сети и легко перегруппировывать устройства в них. Иными словами, коммутаторы позволяют создавать виртуальные сети.

Маршрутизация – процесс определения маршрута данных в сетях связи. Маршрутизация в компьютерных сетях выполняется специальными программно-аппаратными средствами – маршрутизаторами. Маршрутизируемые протоколы определяют формат заголовков пакетов, самое важное в этих заголовках, конечно, адрес назначения и адрес отправителя. У маршрутизатора есть таблица маршрутизации, в которой указано куда отправлять пакет, если адрес принадлежит определенному диапазону адресов (это может быть как другая рабочая станция, так и шлюх или другой маршрутизатор, чтобы передать пакет другой подсети).

Маршрутизация основывается на топологии компьютерной сети. Простейший вид компьютерной сети, при котором два компьютера соединяются между собой, напрямую называется точка-точка. Часто используется в случаях, когда необходимо быстро передать информацию с одного компьютера, например ноутбука, на другой. Если две точки находятся в непосредственной близости, то связь обычно осуществляется через RS232 или подобный протокол (см. Нуль-модемное соединение). При соединении на расстоянии используются модемы.

Другой тип соединения – точка-многоточка, при котором осуществляется подключение вида «один-ко-многим», предоставляя набор соединений от одного абонента с множеством других. В настоящее время этот термин в беспроводной связи относится к обозначению фиксированной беспроводной передачи данных для интернет- или голосового IP-трафика по радио-каналу или микроволновым частотам в гигагерцовом диапазоне частот. Если в сети каждое устройство поддерживает тип соединения точка-многоточка, то говорят о соединении «все-ко-всем»

Городская телефонная сеть – это совокупность линейных и станционных сооружений, обслуживающая примерно 8-10 тысяч абонентов. Для оптимизации целесообразно переходить на районированное построение сети. В этом случае территория города делится на районы, в каждом из которых сооружается одна районная автоматическая телефонная станция (РАТС), к которой подключаются абоненты этого района. РАТС связываются между собой соединительными линиями

в общем случае по принципу «все-ко-всем».

Путь от хоста до шлюза (или от шлюза до шлюза), по которому пересылаются пакеты, принято назвать маршрутом. Выделяют пять типов маршрутов:

- До хоста (если он в той локальной сети, в которой находится маршрутизатор);
- К сети (то есть до шлюза, который передаст данные нужному хосту другой локальной сети или следующему шлюзу);
- По умолчанию (маршрут для сбоев, если по како-либо причине адресат неизвестен);
- Циклический (по умолчанию для всех пакетов локальной сети);
- Оповещения (по умолчанию для всех пакетов оповещения, то есть необходимых к доставке всем узлам).

Каждой подсети, у которой есть IP-адрес, сразу присваиваются два маршрута оповещения (один – адресу подсети, а другой – адресу оповещения подсети).

Самих принципов маршрутизации выделяют два:

- Статическая маршрутизация (обслуживание таблиц маршрутизации вручную).
- Динамическая маршрутизация (автоматическое обслуживание таблиц маршрутизации специальными программами – демонами).

В TCP/IP существует два демона, поддерживающих динамическую маршрутизацию: routed и gated. Демон gated поддерживает следующие протоколы:

- RIP и RIPng (протоколы информации о маршрутизации),
- EGP, BGP и BGP4+ (протоколы внешних и граничных шлюзов),
- OSPF (протокол кратчайшего пути),
- SNMP (протокол управления сетью),
- прочие.

Демон routed поддерживает только протокол информации о маршрутизации.

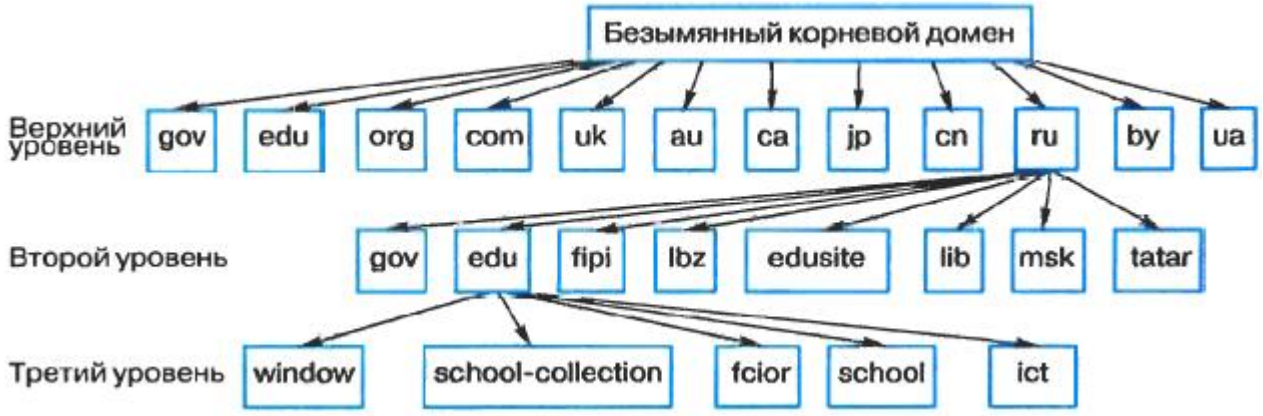
Если демон маршрутизации активен, его задача – оповещать о маршрутизации данной локальной сети шлюзы и хосты, составляющие ее. Делает он это по предустановленному расписанию. И сам так же получает подобные оповещения от прочих демонов своей сети. Также есть и пассивный режим работы демонов маршрутизации. Тогда они только слушают каналы и получают такие сообщения о маршрутизации от прочих демонов, не рассылая свои обновления в ответ.

3.7.Адресация устройств в сети

Как было сказано ранее, все устройства, подключенные к сети, получают свой IP-адрес – уникальный 32-битный идентификатор, по которому узлы сети узнают друг друга. Для удобства использования принято разбивать его на четыре группы по восемь бит. Такую группу называют октетом. Чтобы не возиться с длинными битовыми последовательностями, каждый октет переводят в десятичное число. таким образом можно говорить об адресах от 0.0.0.0 до 255.255.255.255.

Помимо такой адресации узлов сети есть и другая иерархическая система – доменная система имен. Домены верхнего уровня бывают двух видов: административные (трёхбуквенный код для организаций определённого типа) и географические (двухбуквенный код для каждой страны). Целиком доменное им собирается из имени самого домена и имён, всех доменов, в которые он входит, последовательно с разделением через точку.

Рисунок 18. Иерархическая структура доменных имен



Но все же вернемся к IP-адресации. В самой структуре IP-адреса есть две части: адрес сети и номер узла в сети. Стек протоколов TCP/IP объединяет несколько физических сетей в одну большую логическую сеть. Все составляющие сети называют подсетями. И как в них непосредственно организована адресация по большому счету не важно. Протоколы как рас призваны нивелировать все различия локальных устройств, чтобы передача данных была к ним нечувствительна. Ранее адресация велась по трем классам: A, B и C. Отличались сети разных классов по размерам, так как от класса зависела длина той части адреса, что отвечал за сеть и номер соответственно.

Адрес класса A обеспечивал самые крупные сети, на адрес самой сети приходился только первый октет, а локальный адрес узла сети мог быть пронумерован аж 24-мя битами. То есть такие сети могут связывать до 16 777 214 узлов. Самих же сетей может быть 126, поскольку старший бит в адресе указывает на класс (в классе A – всегда 0) и оставшиеся 7 бит первого октета позволят пронумеровать $2^7 - 2$ сетей (напомним, что два адреса отдаются на локальный циклический адрес и адрес оповещения).

Например, 125.13.73.15 – адрес класса A, так как в его двоичном представлении (011111101.00001101.01001001.00001111) старший бит равен 0. При этом первый октет (125) указывает на адрес сети (125.0.0.0), а оставшиеся биты показывают номер узла в ней (11010100100100001111 = 870 671).

Адрес класса B предлагает первые два октета для нумерации сети, и оставшиеся два – на нумерацию узлов в ней. В этом классе уже первые два бита отдаются на указание класса (в классе B всегда 10), остальные 14 бит позволяют идентифицировать $2^{14} - 2$ сетей (таким образом выделяется 16 384 возможных адресов сети). Для адресации узлов каждой такой сети выделяется уже $2^{16} - 2$ адресов.

Примером адреса такого класса будет 157.13.73.15. В двоичном представлении (10011101.00001101.01001001.00001111) два старших бита указывают на класс адресации (B), адрес сети в таком случае 157.13.0.0, а номер узла в ней с таким адресом – (100100100001111) 9 351.

Адрес класса C позволяет соединять самые немногочисленные сети, так как первые три октета предназначены для адреса самой сети и только последний октет нумерует узлы в ней. Таким образом на каждую сеть выделено только $2^8 - 2$ адресов для узлов. Для указания класса в таких адресах старшие три бита устанавливают равными 1-1-0. В итоге $2^{21} - 2$ таких сетей можно поименовать.

В качестве примера адреса такого класса приведем 221.13.73.15. в двоичном представлении (11011101.00001101.01001001.00001111) первые три бита показывают класс, адресом сети является 221.13.75.0, а номер узла в ней – 15.

Существует также и класс D (на который указывают четыре старших бита 1-1-1-0). Однако

система классов уже считается устаревшей и на деле не используется. Вместо неё в бесклассовой системе для определения адреса сети и номера узла прибегают к маске подсети.

Маска сети – 32-битная последовательность из подряд идущих n единиц и следом за ними 32-n нулей, по которой определяется адрес сети. Путем побитового умножения двоичного представления маски и IP-адреса. Маску также принято разделять на четыре октета и записывать в десятичном виде. Разберемся на примере.

Пусть IP-адрес: 192.168.11.2, а маска этой подсети: 255.255.252.0. тогда чтобы узнать адрес сети, нужно расписать двоичные представления и перемножить их побитово:

11000000.10101000.00001011.00000010 – двоичное представление IP-адреса
11111111.11111111.11111100.00000000 – двоичное представление маски
11000000.10101000.00001000.00000000 – результат умножения – адрес сети

Таким образом, адрес сети: 192.168.8.0.

Нетрудно заметить, что для адреса сети были взять из IP-адреса хоста только те биты, что совпадали с установленными битами маски, остальные же были сброшены.

Количество нулей в маске помогает понять, какое количество хостов может быть в подсети. Так, если в маске n единиц и 32-n нулей, то можно соединить в подсеть $2^{32-n} - 2$ рабочих станций. Рассмотрим такую задачу:

Предложите минимальную маску подсети для соединения 44 вычислительных устройств.

Каждая маска может выделить подсеть для узлов в количестве «степень двойки минус два». Таким образом нужно найти минимальную степень двойки, которая удовлетворит уравнению: $2^t - 2 \geq 44 > 2^{t-1} - 2$. Это 6, так как

$$2^6 - 2 = 62 \geq 44 > 2^5 - 2 = 30$$

Значит, правильная маска содержит $32 - 6 = 26$ единиц и 6 нулей:

$$11111111.11111111.11111111.11000000 \rightarrow 255.255.255.192$$

При установлении сеанса связи узел сети отправляет сообщение, коммуникационная система определяет, находится ли получатель в той же подсети (то есть можно напрямую строить маршрут только в подсети) или он является внешним для подсети (и тогда нужно прибегнуть к помощи маршрутизаторов и строить маршрут до шлюза). Чтобы это определить система сравнивает адреса сетей отправителя и получателя с помощью маски.

3.8.Межсетевые экраны

Межсетевой экран – это специальная программа, которая фильтрует сетевой трафик, проходящий через нее, по правилам, установленным администратором безопасности. Если трафик не подходит под шаблон допустимого, межсетевые экраны не пропускают его в сеть. Сами экраны могут быть организованы по одному из двух следующих принципов:

- «Что явно не запрещено, то разрешено» – то есть правила говорят о том, какой трафик точно является вредным для сети, и если всем правилам он не удовлетворяет, тогда его можно пускать в сеть;
- «Что явно не разрешено, то запрещено» – то есть правила говорят о том, какой трафик точно является допустимым, и если всем правилам он не удовлетворяет, тогда его можно нельзя.

Очевидно, что второй подход является более надежным, но трудозатратным и сложно администрируемым, так как в этом случае специалисту, настраивающему межсетевой экран

придется постоянно реагировать на изменения запросов владельцев сети и регулировать настройки межсетевого экрана.

Организовать фильтрацию пакетов можно на разных уровнях коммуникационной системы единственный уровень, на которм не производят фильтрацию, конечно, физический, потому что фильтруют именно содержимое полей заголовка пакета, а на этом уровне речь о пакетах уже не ведется, там существуют только суровые битовые последовательности. Ниже приведем схему соответствия различных межсетевых экранов с уровнями модели OSI.

<table>
<tr><td>Прикладной</td><td colspan="2">Посредники прикладного уровня</td><td rowspan="4">Инспекторы состояния</td></tr>
<tr><td>Представления</td><td></td><td></td></tr>
<tr><td>Сеансовый</td><td></td><td rowspan="3">Шлюзы сеансового уровня</td></tr>
<tr><td>Транспортный</td><td rowspan="2">Пакетные фильтры</td></tr>
<tr><td>Сетевой</td><td></td></tr>
<tr><td>Канальный</td><td colspan="2">Управляемые коммутаторы</td><td></td></tr>
<tr><td>Физический</td><td></td><td></td><td></td></tr>
</table>

Рассмотрим немного подробнее характеристики работы этих межсетевых экранов.

Управляемые коммутаторы.

Зона фильтрации: трафик между разными сетями или узлами одной сети.

Принцип действия: фильтрация полей заголовка.

Преимущества: обрабатывают трафик с большой скоростью, что положительно сказывается на скорости передачи данных по сети. К тому же их установка и эксплуатация сравнительно дешево обходится.

Недостатки: не фильтруют внешний трафик из интернета (это протоколы более высокого уровня, поэтому их заголовки помещены в тело пакетов канальных протоколов).

Пакетные фильтры.

Зона фильтрации: трафик на границе с внешней сетью.

Принцип действия: фильтрация полей заголовка.

Преимущества: обрабатывают трафик с большой скоростью, что положительно сказывается на скорости передачи данных по сети. Появились одними из первых межсетевых экранов. Являются самыми распространенными фильтрами.

Шлюзы сеансового уровня.

Зона фильтрации: трафик на границе с рабочей станцией.

Принцип действия: шлюз сеансового уровня перехватывает все входящие пакеты для узла сети и проверяет их на допустимость по установленным правилам для конкретного установленного соединения. Таким образом, сама рабочая станция получает уже только корректный трафик от внешних агентов.

Преимущества: этот принцип действия позволяет защититься от многих атак на систему, например от разведки топологии (когда нарушитель системы безопасности пытается разузнать, как именно устроена сеть для дальнейшей разработки плана атаки) или атак типа Dos/DDos (когда компьютер перегружается большим количеством ненужных запросов, которые должны устроить перегрузку системы).

Недостатки: с таким принципом действия неизменно будет падать скорость коммуникации.

Посредники прикладного уровня.

Зона фильтрации: трафик между двумя узлами сети

Принцип действия: посредник прикладного уровня представляет собой несколько приложений-посредников, каждое из которых отвечает за свой протокол. Фильтруются как обычно поля

заголовков, но с учетом конкретного контекста данной коммуникации.

Преимущества: их можно настроить с большей точностью под целевые коммуникации организации.

Недостатки: с таким принципом действия неизменно будет падать скорость коммуникации. К тому же раз их настройка происходит под конкретную специфику данной сети, то они весьма не просты в установке, настройке и эксплуатации.

Инспекторы состояния.

Зона фильтрации: трафик между разными сетями или узлами одной сети.

Принцип действия: тут работают все идеи вышеназванных межсетевых экранов.

Преимущества: обрабатывают трафик с большой скоростью, что положительно сказывается на скорости передачи данных по сети. Не вмешиваются в сам процесс коммуникации и установления связи.

Недостатки: плохо защищены и дорогостоящи в установке и эксплуатации.

3.9.Вредоносные программы

Этот параграф посвящен конкретным категориям программных продуктов, приносящих вред вашим компьютерам. Компьютерные вирусы существуют довольно давно, и почти все они распространились через Интернет или его предшественников. Большинство вирусов были разработаны для кражи пользовательской информации, вычислительной мощности или отключения системы в целом.

Компьютерный вирус – это вредоносный фрагмент компьютерного кода, который имеет на самом деле простую задачу: распространяться с устройства на устройство, прятаться и запускаться. Множество вредоносных программ – это самокопирующиеся угрозы, которые обычно предназначены для повреждения устройства или кражи данных. Некоторые компьютерные вирусы запрограммированы так, чтобы нанести вред вашему компьютеру, повреждая программы, удаляя файлы или форматируя жесткий диск. Другие просто копируют себя или наводняют сеть трафиком, делая невозможным выполнение каких-либо действий в Интернете. Даже менее вредоносные компьютерные вирусы могут значительно нарушить производительность вашей системы, истощая память компьютера и вызывая частые его сбои. Даже если вы будете осторожны, вы можете подхватить компьютерные вирусы, производя обычные действия в Интернете, например:

- Обмен музыкой, файлами или фотографиями с другими пользователями;
- Посещение зараженного веб-сайта;
- Открытие спама или вложения в электронном письме;
- Скачивание бесплатных игр, панелей инструментов, медиаплееров и других системных утилит;
- Установка основных программных приложений без тщательного изучения лицензионных соглашений.

Подумайте о биологическом вирусе - вирусе, от которого вы заболеете. Это постоянно неприятно, мешает вам нормально функционировать и часто требует чего-то мощного, чтобы избавиться от него. Компьютерный вирус очень похож. Компьютерные вирусы, разработанные для непрерывной репликации, заражают ваши программы и файлы, изменяя способ работы вашего компьютера или полностью останавливая его работу.

Вирусы могут распространяться несколькими способами, в том числе через сети, диски, вложения электронной почты или внешние устройства хранения, такие как USB-накопители. Поскольку соединения между устройствами когда-то были гораздо более ограниченными, чем сегодня, ранние компьютерные вирусы обычно распространялись через зараженные дискеты.

Сегодня связи между устройствами, подключенными к Интернету, являются общими, что дает широкие возможности для распространения вирусов. По данным Агентства по кибербезопасности и безопасности инфраструктуры США, зараженные вложения электронной почты являются наиболее распространенным средством распространения компьютерных вирусов. Большинство компьютерных вирусов, но не все, требуют от пользователя действий, например включения «макросов» или щелчка по ссылке для распространения.

Ваш компьютер, вероятно, заражен, если вы улавливаете какие-либо из этих симптомов вредоносного ПО:

- Низкая производительность компьютера;
- Неустойчивое поведение компьютера;
- Необъяснимая потеря данных;
- Частые сбои компьютера.

Первый компьютерный вирус, названный «Creeper system», был экспериментальным самовоспроизводящимся вирусом, выпущенным в 1971 году. Он заполнял жесткий диск до тех пор, пока компьютер не переставал работать. Этот вирус был создан в Соединенных Штатах Америки в научно-исследовательской компании «BBN technologies».

Первым компьютерным вирусом для MS-DOS был "Brain", выпущенный в 1986 году. Он перезаписывал загрузочный сектор на дискете и предотвращал загрузку компьютера. Он был написан двумя братьями из Пакистана и изначально проектировался вовсе не во вред, а как защита от копирования.

«Моррис» был первым компьютерным вирусом, который широко распространился в «дикой» природе в 1988 году. Он был написан Робертом Моррисом, аспирантом Корнельского университета, который планировал с его помощью определить размеры Интернета. Его подход использовал дыры в безопасности в sendmail (один из первых программ-агентов передачи почты) и других приложениях Unix, а также слабые пароли, но из-за ошибки программирования он распространился слишком быстро и начал мешать нормальной работе компьютеров. За 15 часов он заразил около 15 000 компьютеров, на которых тогда была большая часть Интернета.

С тех пор было введено много новых вирусов, и эта тенденция с каждым годом растет в геометрической прогрессии. Ниже приведены некоторые из наиболее известных или значимых вирусов, рост которых зависит от роли информационных технологий в обществе.

В 1991 году вирус «Микеланджело» был впервые обнаружен в Австралии. Он бездействовал до 6 марта каждого года, а затем перезаписывал первую сотню секторов на устройствах хранения нулями, предотвращая загрузку компьютера. Сообщалось о заражении только 20 000 компьютеров.

В 1998 году был выпущен CIH. Он заразил около 60 миллионов компьютеров и нанес значительный ущерб, перезаписав важные системные файлы. Его написал тайваньский студент.

В 1999 году на экраны вышла «Мелисса». Это был первый широко распространенный макровирус Word. Он распространялся по электронной почте и автоматически рассылался первым 50 людям в адресной книге Outlook. Это не повредило компьютеру, поскольку он рассылал пароли для некоторых эротических сайтов, для которых требовалось членство. Это вызвало такой большой почтовый трафик, что привело к сбою почтовых серверов.

2000 год был годом "iloveyou". Опять же, он пришел по электронной почте, но разослал себя всем контактам. Он также перезаписывал офисные файлы, изображения и аудиофайлы. Вирус пришел с Филиппин и менее чем за 10 дней заразил более 50 миллионов компьютеров. Тогда большинство компаний решили отключить свои почтовые серверы, чтобы остановить распространение вируса.

С 2000 года было выпущено так много новых вирусов, которые нанесли ущерб миру в целом, что трудно перечислить самые печально известные: «Анна Курникова», Code Red, Nimba, Beast, SQL Slammer, Blaster, Sobig, Sober, MyDoom, Netsky, Zeus, Conficker, Stuxnet, CryptoLocker, Locky, Mirai

и WannaCry.

В 2013 году с появлением вируса CryptoLocker появилась новая форма кибер-вымогателей. Это разработанная программа-вымогатель, которая распространялась через заражённые вложения электронной почты и заражённые сайты. На компьютере она шифровала файлы, используя шифр RSA (ключ расшифровки е сохранялся на сервере злоумышленника), и предлагала владельцу компьютера перечислить определенную сумму денег за восстановление системы.

Было много новых версий этого вируса, включая Locky и WannaCry, а также Petya (не последняя версия). Вирус CryptoLocker в своей исходной версии заразил около полумиллиона компьютеров. Некоторые из этих клонов, например TorrentLocker или CryptoWall, были специально разработаны для компьютеров в Австралии. WannaCry и NotPetya использовали брешь в безопасности Windows, который использует для доступа к файлам по сети протокол SMB. Эта дыра в безопасности, названная EternalBlue, была обнародована хакерской группой под названием «Shadow Brokers», которая украла ее у Агентства национальной безопасности США (NSA). Хотя Microsoft выпустила исправление для этой уязвимости в марте 2017 года, количество систем во всем мире, основанных на устаревшем или неподдерживаемом программном обеспечении (или еще не применявших последние обновления), позволило WannaCry прочно закрепиться за счет фишинговых атак по электронной почте. WannaCry заразил около 200 000 компьютеров в 150 странах, прежде чем был обнаружен «выключатель убийства», который остановил распространение вируса.

Совсем недавно NotPetya воспользовался той же дырой в безопасности. Однако он не был доставлен по электронной почте и поэтому имел ограниченный охват. Сначала предполагалось, что этот вирус может быть обновленной версией Petya, вымогателя типа CryptoLocker. Фактически NotPetya распространялся как обновленная версия украинского пакета налогового учета под названием MeDoc, а оттуда он начал распространяться по внутренним сетям транснациональных компаний с офисами в Украине. Он шифрует все файлы на компьютере, а также главную таблицу файлов на жестком диске, предотвращая загрузку компьютера. NotPetya имел очень простую платежную систему по сравнению с другими вирусами-вымогателями. Это привело к общему мнению, что «петинская» часть вируса была просто приманкой, и восстановление файлов оказалось невозможным.

По мере появления новых вирусов производители антивирусного программного обеспечения применяют новые инструменты для борьбы с ними. Однако это постоянная игра в кошки-мышки, так как злоумышленники также не сидят на месте и постоянно ищут новые уязвимости автоматизированных систем.

Большинство вирусов-вымогателей невозможно обнаружить с помощью классического антивируса, поэтому компании по кибербезопасности начали проводить мониторинг поведения для их обнаружения. Однако это лишь вопрос времени, пока не появится новый вирус, который найдет способ обойти каждый новый метод обнаружения, и весь процесс не начнется заново.

Когда риски постоянно меняются, лучшие шаги, которые помогут вам оставаться в безопасности, остаются неизменными – постоянная бдительность в борьбе с фишинговыми сообщениями электронной почты и мошенническими веб-сайтами как наиболее распространенными способами заражения:

- Не открывайте электронные письма и вложения к ним, если вы не на 100% уверены, что они законны.
- Не нажимайте ссылки в электронных письмах или их прикрепленных файлах, если вы не ожидали их получить. Помните, что учетные записи электронной почты могут быть подделаны или взломаны, поэтому, хотя может показаться, что сообщение пришло из законного источника, если содержимое не соответствует вашим ожиданиям от этого отправителя, оно может быть ненадежным.
- Регулярно обновляйте свой компьютер, устанавливая последние обновления программного

обеспечения и исправления безопасности.

• Проверяйте орфографические или грамматические ошибки – в том числе в URL-адресах веб-сайтов, которые вы посещаете, а также в теле электронных писем. Например, ошибочно приняв ofice.com за Microsoft office.com, вы попадете на сайт известного вредоносного ПО.

• Обязательно сообщайте о любых подозрительных электронных письмах или необычном поведении системы как можно скорее. Рекомендуется переслать подозрительное электронное письмо в качестве вложения в службу поддержки для расследования.

Антивирусы добились больших успехов в обнаружении и предотвращении распространения компьютерных вирусов. Однако, когда устройство действительно заражено, лучшим вариантом для его удаления по-прежнему является установка антивирусного решения. После установки большая часть программного обеспечения будет выполнять «сканирование» на наличие вредоносной программы. После обнаружения антивирус предложит варианты его удаления. Если это невозможно сделать автоматически, некоторые поставщики средств безопасности предлагают техническую помощь в удалении вируса бесплатно. Классификация антивирусов приведена ниже.

Сканеры (или фаги, или полифаги).

Сканеры регулярно проверяют файлы, сектора и системную память в попытке найти действия, подходящие под сигнатуру определенного вируса, сохраненного в их базе данных. Сигнатурой (или маской) вируса называют стандартные действия той или иной вредоносной программы или эффект, который эти действия оказывают на систему.

Очевидным минусом такого подхода становится то факт, что злоумышленники постоянно также разрабатывают новые вирусы, против которых сканеры бессильны, так как не знают их сигнатур.

Говоря о принципе действия сканеров, их можно разбить на два класса: резидентные (которые еще называют мониторами) и нерезидентные. Мониторы сканируют все действия сразу же при их совершении на устройстве, чем и обеспечивают большую защиту и сильную нагрузку на ресурсы. Нерезидентные сканеры выполняют свою работу только по прямому запросу, что не позволяет им своевременно реагировать на проявление странного поведения со стороны вредоносной программы, обнаружить ее действия они смогут лишь тогда, когда их запустит пользователь.

CRC сканеры

CRC – это посчитанное по специальной формуле значение для файла, позволяющее его идентифицировать. Главные особенности CRC заключаются в том, что при малейшем изменении файла новое рассчитанное значение CRC должно это отразить. Использовать это значение удобнее, чем, например, копию для сравнения состояний, так как CRC намного меньше файла.

CRC могут использоваться для различных целей. Сканер вирусов может использовать их для пропуска файлов, которые были просканированы, чтобы ускорить сканирование.

Недостатки:

- CRC сканер должен запускаться при каждом обновлении файла, что затормаживает работу системы, особенно если сканер направлен на часто перезаписываемый файл;
- CRC сканер не может поймать вирус в момент его появления в системе, а делает это только через некоторое время после проверок и уже плотного оседания вредоносной программы в компьютере;
- CRC сканер не может определить вирус в новых файлах (в электронной почте, на дискетах, в файлах, восстановленных из резервной копии или при распаковке файлов из архива), для которых он заранее не рассчитывал CRC значение и не сохранял его в своей базе;
- наличие коллизий CRC алгоритма.

Некоторые файлы нельзя проверить с помощью CRC, поскольку они постоянно меняются. CRC можно использовать для идентификации известных хороших файлов и заведомо плохих файлов. Программа безопасности может проверять свой CRC при запуске, чтобы убедиться, что она также не была взломана. CRC – это быстрый способ проверить правильность записи компакт-диска, если

вы использовали файл образа.

На практике CRC мало используются для обеспечения безопасности, потому что многие файлы могут иметь идентичные CRC-значения. Эта ситуация и называется коллизией или конфликтом. Придумать алгоритм расчёта таких образов файла совсем без коллизий – не выйдет. Это легко показывается математически.

Другой метод создания таких образов файлов – посчитать для них хеш-значение (тоже некоторое число, рассчитанное по определенному алгоритму). Хеши MD5 используются так же, как CRC, но имеют меньшую частоту конфликтов. Исследователи продемонстрировали надежные способы увеличения частоты конфликтов для MD5, поэтому SHA1 и SHA2, особенно SHA2, становятся все более распространенными способами идентификации файлов. Эти технологии имеют чрезвычайно низкую частоту конфликтов, и злоумышленнику намного сложнее создать файл с идентичным хешем.

Блокираторы

Блокировщик поведения – это тип программы, которая предотвращает выполнение определенных действий, потенциально похожих на запросы вредоносного программного обеспечения. Под «подозрительным на вирус» понимаются обращения и попытки открытия для записи исполняемых файлов, записи в загрузочные секторы дисков, также подозрительным кажется попытка от программы оставаться резидентной и прочее. Блокировщик поведения может препятствовать записи программы в реестр, загрузочный сектор или файлы.

Иногда технологии блокировки поведения встраиваются в программы, у которых есть и другие возможности. Вирусы загрузочного сектора были настоящей проблемой, и они практически исчезли в Microsoft с этой технологией. Тем не менее, у блокираторов поведения тоже есть свои проблемы. Блокаторы поведения могут быть отличным уровнем защиты, но в умеренных количествах. Также существуют аппаратные и программные средства блокировки поведения. Некоторые BIOS содержат перемычку, которую необходимо удалить, чтобы перепрограммировать BIOS, или настройку микропрограммы, которую необходимо изменить для записи в загрузочный сектор или загрузки с CDROM.

Преимуществом блокировщиков поведения является их способность обнаруживать и даже останавливать вирус на ранней стадии его размножения.

Недостатки:

- наличие способов защиты вируса от блокираторов (технологии вирусов-невидимок);
- большое количество ложных срабатываний на доверенные программы.

Иммунизаторы

Иммунизаторы пытаются предотвратить заражение вирусом. Эта технология используется не часто, поскольку ее возможности ограничены и ее необходимо постоянно обновлять. По принципу действия иммунизаторы делятся на два класса: те, что только информируют о заражении, и те, что блокируют заражение.

Иммунизаторы, информирующие о заражении, записываются в конец файла, как это делает файловый вирус, и при очередном обращении к этому файлу проверяют его на внесение подозрительных изменений.

Некоторые вирусы записывают некоторое известное им значение в реестр при заражении компьютера, чтобы при повторном запуске не тратить ресурсы на повторное заражение. Компьютер можно иммунизировать, просто добавив такое значение в реестр до первого заражения вирусом. Тогда при первом запуске вредоносной программы, она сразу считает это значение в реестре и не станет заражать компьютер. Также некоторые вирусы оставляют подобный маркер в некоторых файлах, и, если маркер был помещен в файл заранее, вирус не заражает его. Этим предварительным внесением соответствующих меток и занимаются иммунизаторы, которые блокируют заражение.

Недостатки:

- не способны предотвратить заражение вирусом-невидимкой;
- не являются универсальным средством, так как вирусы постоянно модифицируются и

сложно заранее знать, какую метку оставит для себя новый вирус.

Раздел 4. Кодирование информации

4.1. Кодирование информации

Кодирование – это процесс преобразования данных из формы, удобной для непосредственного использования человеком, в форму, удобную для автоматической обработки данных (то есть их хранения, передачи и сохранения от несанкционированного доступа). Это один из самых важных инструментов обеспечения информационной безопасности.

Существуют следующие 4 вида кодирования.

Кодирование источника (или сжимающее кодирование)

Сжатие данных – это преобразование данных по некоторому алгоритму, производимое для уменьшения занимаемого ими объёма. У всякой не информации, которая не является случайным набором символов, обязательно есть избыточность. Самый простой пример – повторяющиеся части текста. Именно за счет ликвидации избыточности можно уменьшать объем данных так, чтобы потом можно было восстановить их обратно. Цель такого преобразования очевидно – экономия важнейших ресурсов – памяти во время хранения данных и времен во время передачи их по каналу связи. Процесс восстановления сжатых данных называют декомпрессией или распаковкой.

Под избыточностью более формально в теории информации принято понимать превышение количества информации, используемой для передачи или хранения сообщения, над его информационной энтропией (с этим термином мы познакомились в предыдущих главах).

Также технически избыточностью можно признать неравномерное частотное распределение символов алфавита источника, которое обычно наблюдается в естественных текстах. Для сокращения объема памяти в таком случае можно прибегнуть к специальному виду кодирования – энтропийному. Главной его идеей является как раз закодировать частые элементы короткими битовыми последовательностями, а редкие – длинными. Тогда частотное распределение станет более равномерным, и очевидно, данные будут занимать меньше памяти, по сравнению с тем случаем, когда кодирование производилось бы без учета частоты появления этих элементов.

Сжатие данных можно производить двумя способами: с потерями и без потерь. Пусть вас не смущает допустимость потерь вообще. Главное – не потерять информационной ценности сообщения, чтобы его можно было восстановить и воспринимать без особых упущений. Однако от потери одного бита на незначащей позиции никто сильно не пострадает. Это хорошо заметно на примере обработки графической информации. Сжатие картинки (в разумных пределах) не влияет на наше восприятие, контуры останутся на своих местах, цвета будут те же. Конечно, если ее не сжали настолько, что от былого изображения в ней ничего не осталось.

Канальное кодирование (или помехоустойчивое кодирование)

Данные преобразования принято производить над информацией перед отправкой ее по каналу связи (отсюда и название). Делают это с целью повысить помехоустойчивость. Говоря о каналах в предыдущих главах, мы уже упоминали, что даже без явного нарушителя информация может потерять свою целостность при передаче ее по каналу связи, так как физика играет не в нашу пользу – помехи среды распространения будут вносить свои коррективы в передаваемые данные.

Главным образом, задача помехоустойчивого кода – добавить к информационной последовательности бит некоторые дополнительные – служебные, по которым можно будет следить за состоянием данных и при получении распознать хотя бы наличие помех. Конечно, такие коды не всемогущи и у них есть ограничения на свои действия, но подробнее о характеристиках будет рассказано в следующих параграфах.

В отличие от сжимающих кодов, которые нужны для уменьшения объема памяти, занимаемого данными, помехоустойчивые коды неизбежно будут увеличивать этот объем, поскольку невозможно на биты, отвечающие за информацию, назначить еще дополнительно и функцию отслеживания своего состояния. Для новой задачи неизбежно придется добавить и новые биты.

Поэтому эти два вида кодирования, компенсируют эффект друг друга.

По принципам действия канальные коды принято разделять на коды обнаружения и коды прямой коррекции.

Обнаруживающие коды способны лишь сигнализировать с некоторой точностью о наличии помех в последовательности данных, но они никак не помогут с ней справиться. Чтобы восстановить корректное состояние информации (если это кране необходимо и пренебречь этим нельзя), придется запрашивать повторную отправку этого фрагмента данных у отправителя.

Коды прямой коррекции – класс более сильных помехоустойчивых кодов, которые позволяют не только обнаружить, но и исправить ее самостоятельно. Смотрится как чудо! Но конкретный алгоритм будет приведен в параграфах ниже, так что вы убедитесь на деле, что в математике чудеса случаются.

Криптография

Криптография – наука о шифровании, скажет обыватель. Мы же скажем, и не только о нем. Строго говоря, криптографические преобразования данных – это подвид кодирования. Целью такого преобразования, в первую очередь, выступает, конечно, защита конфиденциальности информации, но помимо этого у криптографии есть и другие задачи, одной из которых является и защита целостности.

Физическое кодирование

Физическое кодирование, это способ представления данных в виде каких-либо сигналов, пригодных для среды, в которую должна перейти информация (для хранения или передачи). Например, в виде дискретных уровней амплитуды напряжения, амплитуды тока, амплитуды яркости и прочего, о чем мы вели рассказ в предыдущих главах.

4.2. Параметры оценки кода

В теории кодирования принято выделять две основные проблемы:

- Взаимная однозначность декодирования (то есть, вообще говоря, возможность восстановления преобразованной информации);
- Сложность реализации кода (чем более сложную задачу должен решать код, тем более сложен его алгоритм, что неизбежно пагубно сказывается на ресурсах, таких как время, оперативная память и скорость коммуникации. К тому же алгоритмы должны быть просты и понятны для применения конечным пользователям).

Код является однозначно декодируемым, если любая последовательность символов из алфавита кода (а, в основном, это 0 и 1) разбивается на отдельные слова. Для взаимно однозначного кодирования можно использовать разделители, блочные коды или специально построенный префиксные коды.

Требование префиксности ограничивает множество длин кодовых слов и не даёт возможности выбирать кодовые слова слишком короткими. Существуют разные алгоритмы построения префиксных кодов.

Условие Фано: для того, чтобы сообщение, записанное с помощью неравномерного по длине кода, однозначно раскодировалось, достаточно, чтобы никакой код не был началом другого (более длинного) кода.

Обратное условие Фано также является достаточным условием однозначного декодирования неравномерного кода. В нём требуется, чтобы никакой код не был окончанием другого (более длинного) кода.

Для возможности однозначного декодирования достаточно выполнения одного из условий – или прямого, или обратного.

Заметим, что существуют варианты неравномерного кодирования, для которых оба условия нарушены, и тем не менее они однозначно раскодируемые.

Однозначность восстановления информации, условие Фано. Однозначно декодируемые, префиксные и суффиксные коды

Идея блочных кодов состоит в том, что разбиение двоичной последовательности на отдельные кодовые слова происходит естественным путем отсчитывания блоков фиксированной длины. Не придется определять, где начало и где конец кода каждого символа, если известно, что все символы имеют фиксированную конкретную длину.

Идея разделителей достаточно проста. Для отделения одного кодового слова от другого используется специальная зарезервированная служебная последовательность.

Рассмотрим для начала параметры оценки сжимающих кодов.

При использовании сжатия без потерь возможно полное восстановление исходных данных, сжатие с потерями позволяет восстановить данные с искажениями, обычно несущественными с точки зрения дальнейшего использования восстановленных данных. Сжатие без потерь обычно используется для передачи и хранения текстовых данных, компьютерных программ, реже – для сокращения объёма аудио- и видеоданных, цифровых фотографий и т. п., в случаях, когда искажения недопустимы или нежелательны. Сжатие с потерями, обладающее значительно большей, чем сжатие без потерь, эффективностью, обычно применяется для сокращения объёма аудио- и видеоданных и цифровых фотографий в тех случаях, когда такое сокращение является приоритетным, а полное соответствие исходных и восстановленных данных не требуется.

Важнейшей характеристикой помехоустойчивых кодов является скорость кода

Скоростью кода называют отношение количества информационных символов исходного сообщения к количеству передаваемых символов кодовой последовательности. По данному показателю можно оценивать экономичность кода, что необходимо для понимания, какой алгоритм меньше нагружает канал связи.

4.3. Физическое кодирование

Физическое кодирование, это способ представления данных в виде каких-либо сигналов. Например, в виде дискретных уровней амплитуды напряжения, амплитуды тока, амплитуды яркости и т. п.

Физическое кодирование – представления дискретных сигналов, передаваемых по цифровому каналу связи, с целью передачи данных, представленных в цифровом виде, на расстояние по физическому каналу связи (такому как оптическое волокно, витая пара, коаксиальный кабель, инфракрасному излучению). Физическое кодирование также применяется для записи данных на цифровой носитель.

При физическом кодировании решаются вопросы синхронизации, управления полосой пропускания сигнала, скорость передачи данных и расстояние, на которое необходимо передать

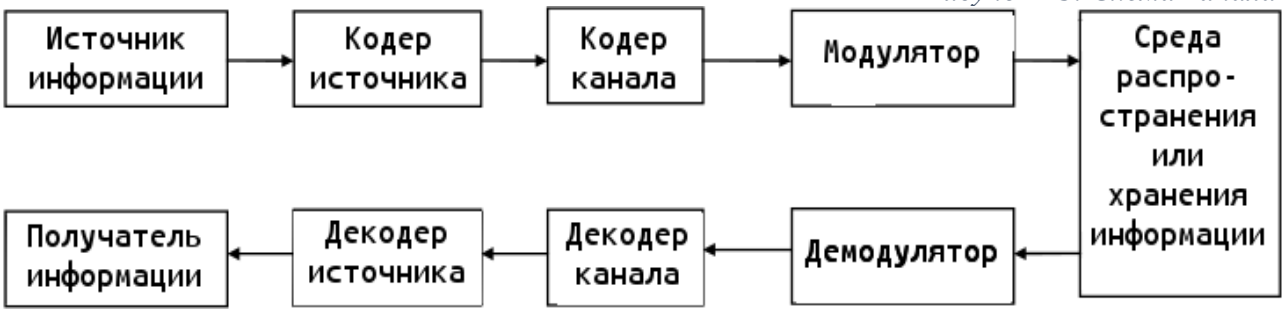
данные. Различают виды передачи дискретных сигналов:

- синхронный способ передачи данных — когда приёмник и передатчик работают синхронно (в один такт);
- асинхронный способ передачи данных — когда приёмник и передатчик работают несинхронно.

4.4. Модуляторы

Для кодирования данных предусмотрены специальные устройства на разных уровнях передачи данных:

Рисунок 19. Схема канала связи



В схеме источником является любой объект вселенной, порождающий сообщения, которые должны быть перемещены в пространстве и времени. Независимо от изначальной физической природы, все подлежащие передаче сообщения обычно преобразуются в форму электрических сигналов, такие сигналы и рассматриваются как выход источника.

Кодер источника представляет информацию в наиболее компактной форме.

Кодер канала обрабатывает информацию для защиты сообщений от помех при передаче по каналу связи или возможных искажений при хранении информации.

Модулятор преобразовывает сообщения, формируемые кодером канала, в сигналы, согласованные с физической природой канала связи или средой накопителя информации.

Среда распространения информации (канал связи) вносит в процесс передачи информации случайный шум, который искажает сообщение и тем самым затрудняет его прочтение.

Модулятор – устройство, изменяющее параметры несущего сигнала в соответствии с изменениями передаваемого (информационного) сигнала. По виду управляемых параметров модуляторы делятся на:

- амплитудные,
- частотные,
- фазовые и др.

Раздел 5. Кодирование источника

5.1. Кодирование источника

Сжатие данных, это алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных. Обратная процедура называется восстановлением данных (распаковкой, декомпрессией). Сжатие основано на устранении избыточности, содержащейся в исходных данных. Простейшим примером избыточности является повторение в тексте фрагментов (например, слов естественного или машинного языка).

Сжатие данных — алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных.

Сжатие основано на устранении избыточности, содержащейся в исходных данных. Простейшим примером избыточности является повторение в тексте фрагментов (например, слов естественного или машинного языка). Подобная избыточность обычно устраняется заменой повторяющейся последовательности ссылкой на уже закодированный фрагмент с указанием его длины.

Другой вид избыточности связан с тем, что некоторые значения в сжимаемых данных встречаются чаще других. Сокращение объёма данных достигается за счёт замены часто встречающихся данных короткими кодовыми словами, а редких — длинными (энтропийное кодирование). Сжатие данных, не обладающих свойством избыточности (например, случайный сигнал или белый шум, зашифрованные сообщения), принципиально невозможно без потерь.

5.2. Алфавитное кодирование

Суть алфавитного кодирования заключается в том, чтобы каждому символу алфавита, которым мы пишем исходное сообщение, сопоставить некую кодирующую последовательность символов (например, последовательность 0 и 1, как это принято при оцифровке). Такую последовательность принято называть кодовым словом, а их совокупность для всех символов начального алфавита – кодом.

В качестве примера алфавитного кодирования можно рассмотреть знаменитый код Морзе. Это способ знакового кодирования, преобразования букв алфавита, цифр, знаков препинания и других символов в двоичные последовательности сигналов, которые может переносить информационный канал: существуют два кодовых элемента: точка (короткий знак) и тире (длинный). За единицу времени принимается длительность одной точки. Тире при передаче имеет «длину» трех точек. Чтобы отличать один знак от другого, пропускается пауза (равная длине точки), чтобы отличать буквы, пропускают тире. Вместо пробела для разделения слов используют паузу длиной в семь точек. Этот популярный метод кодирования используется до сих пор.

Рисунок 20. Код Морзе латинского алфавита

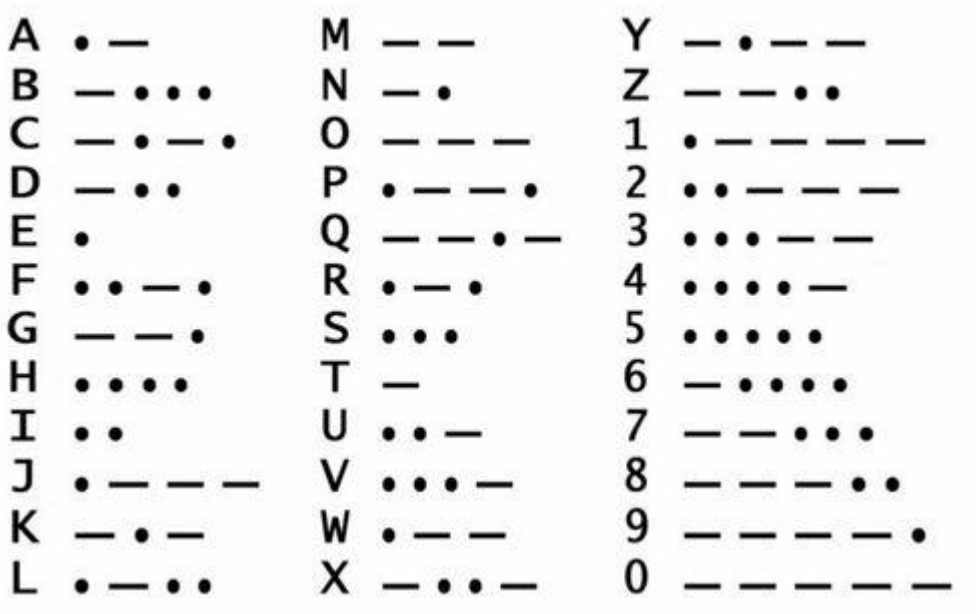


Рисунок 21. Расширенный код Морзе русского алфавита

Как и в любой системе, у морзянки существуют достоинства и недостатки. К достоинствам можно отнести следующие факторы:

– Высокая помехозащищённость при приёме на слух в условиях сильных радиопомех;
– Возможность кодирования вручную;
– Узкая полоса занимаемых частот;
– Запись и воспроизведение сигналов простейшими устройствами.

Недостатки же в азбуке Морзе следующие:

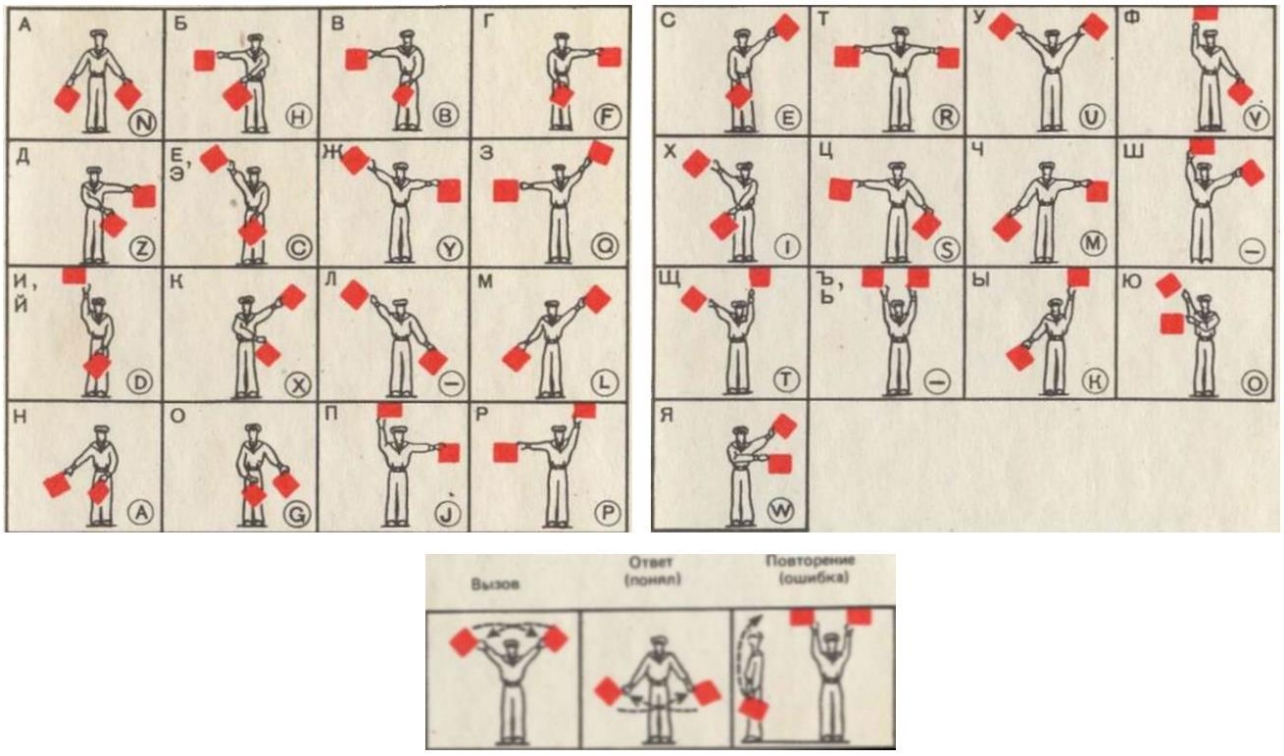
– Неэкономичность, на передачу одного знака кода требуется в среднем 9.5 элементарных посылок;
– Малая пригодность для буквопечатающего приёма (из-за переменной длины кода);
– Низкая скорость телеграфирования (из-за переменной длины кода необходимость длинных пауз между передаваемыми символами).

Другим нестандартным примером алфавитного кодирования является семафорная азбука Макарова. Используется она и в наши дни, в основном на флоте. А нестандартной является, потому что буквы заменяются не на двоичные последовательности, да и вообще не передаются по

цифровому каналу. Каждой букве и условному знаку соответствует определенное положение рук с флажками. А канал используется зрительный. Скорость передачи текста обученным семафорной азбуке сигнальщиком составляет 60-80 знаков в минуту.

Русская семафорная азбука составлена в соответствии с русским алфавитом, включает 29 буквенных и 3 служебных знака. Существует и международная семафорная азбука, использующая те же знаки.

Рисунок 22. Русская семафорная азбука Макарова с сопоставлением знаков латиницы



Более часто употребляемым алфавитным кодированием, которое также использует зрительный канал передачи информации, является дактиль. Или жестовая азбука глухонемых. Это не жестовый язык, в котором жесты обозначают разные слова, а именно алфавитное кодирование, т.е. один жест на одну букву.

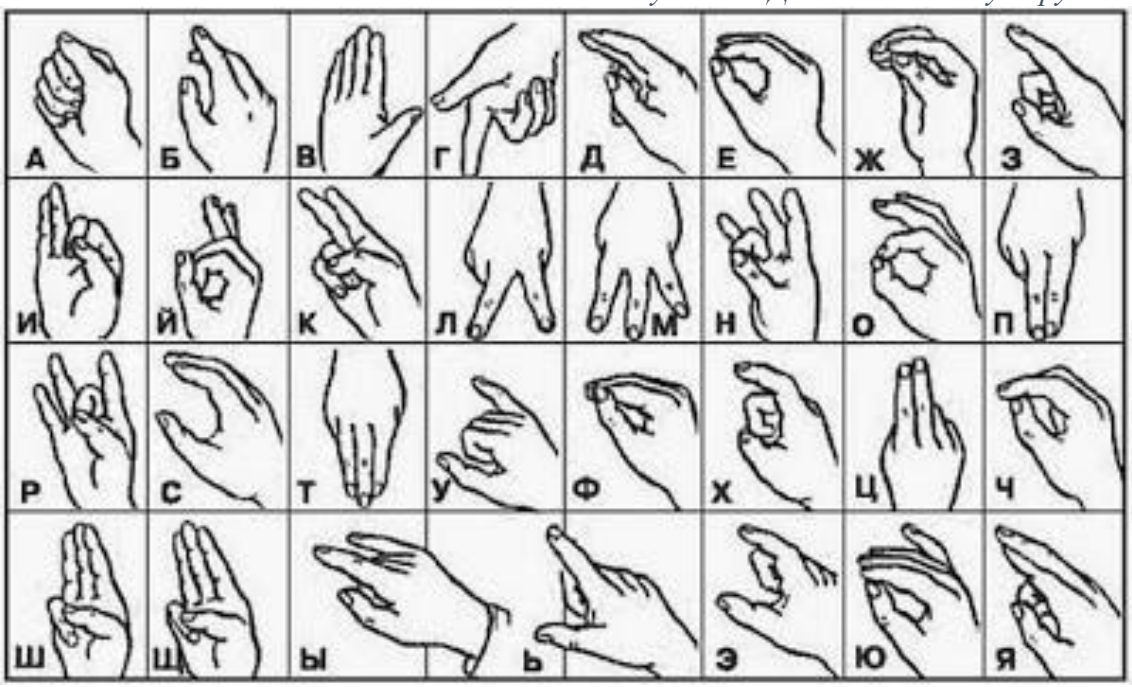
Буквы дактильного алфавита (как мы бы сказали – кодовые слова) называются дактилемы. Это различные положения пальцев, которые воспроизводят знаки, функционально аналогичные буквенному алфавиту; и по очертанию многие из них отдалённо напоминают буквы печатного шрифта (например, «о», «м», «г», «ш»). Каждое положение пальцев при этом означает букву.

Процесс общения на дактиле (дактилология) происходит таким образом, что говорящий показывает буквы на дактиле, а воспринимающий следит за движением руки визуально. Если же зрение человека не позволяет ему видеть знаки, и он воспринимает их осязательно, то такое общение называется дактильно-контактной речью (ДКР). Если же при общении со слепоглухими людьми на ладони другого человека пальцами воспроизводят очертание букв, то это уже не дактилология, а письмо на ладони.

В настоящее время в мире существует более 40 таких алфавитов и систем. Количество знаков в «пальцевом алфавите» зависит от знаков в алфавите языка, хотя не всегда они равны. В России, к примеру, 30 знаков передают 33 буквы кириллицы. В большинстве «пальцевых азбук» используется одна рука. Считается, что это «более удобно», так как вторая рука свободна для других действий.

Однако, есть системы, где используются две руки, например, в британском, австралийском, новозеландском дактилях.

Рисунок 23. Дактильная азбука русского языка



5.3. Однозначность восстановления информации

К основным проблемам теории кодирования относят вопросы взаимной однозначности кодирования (возможность декодирования) и сложности реализации канала связи при заданных условиях (баланс между сложностями двух процессов: кодирования и декодирования).

Код является однозначно декодируемым, если любая последовательность символов из алфавита кода (а, в основном, это 0 и 1) разбивается на отдельные слова. Для взаимно однозначного кодирования можно использовать разделители, блочные коды или специально построенный префиксные коды.

Требование префиксности ограничивает множество длин кодовых слов и не даёт возможности выбирать кодовые слова слишком короткими. Существуют разные алгоритмы построения префиксных кодов.

Условие Фано: для того, чтобы сообщение, записанное с помощью неравномерного по длине кода, однозначно раскодировалось, достаточно, чтобы никакой код не был началом другого (более длинного) кода.

Обратное условие Фано также является достаточным условием однозначного декодирования неравномерного кода. В нём требуется, чтобы никакой код не был окончанием другого (более длинного) кода.

Для возможности однозначного декодирования достаточно выполнения одного из условий – или прямого, или обратного.

Заметим, что существуют варианты неравномерного кодирования, для которых оба условия нарушены, и тем не менее они однозначно раскодируемые.

Идея блочных кодов состоит в том, что разбиение двоичной последовательности на отдельные кодовые слова происходит естественным путем отсчитывания блоков фиксированной длины. Не придется определять, где начало и где конец кода каждого символа, если известно, что все символы имеют фиксированную конкретную длину.

С разделителями мы уже познакомились, рассматривая азбуку Морзе. Для отделения одного кодового слова от другого используется специальная зарезервированная служебная последовательность.

5.4. Стратегии реагирования

На передаваемую информационную последовательность бит всегда будут накладываться помехи. Но как же поступать с ними приемнику? В противном восприятии можно обозначить три стратегии реагирования:

- сброс
- повтор
- восстановление

Проще всего не обращать внимание на эти помехи, конечно, так можно поступать не всегда и только в том случае, если их количество не критично. Несколько ошибочных бит в картинке, например, могут вообще быть незаметны глазу.

Если же такая стратегия не подходит, можно запросить повторную отправку у отправителя. Это вновь займет канал связи, зато информация будет получена.

Если же предварительно на сообщение был наложен специальный канальный код, который как раз нужен для защиты от помех, то можно восстановить самостоятельно полученную информацию. Этот метод наиболее эффективен, однако требует дополнительных ресурсов для реализации. К тому же помехоустойчивые коды не могут творить чудеса, у них также есть пределы: если будет утрачена большая часть информации, такие коды не спасут.

При использовании сжатия без потерь возможно полное восстановление исходных данных, сжатие с потерями позволяет восстановить данные с искажениями, обычно несущественными с точки зрения дальнейшего использования восстановленных данных. Сжатие без потерь обычно используется для передачи и хранения текстовых данных, компьютерных программ, реже — для сокращения объёма аудио- и видеоданных, цифровых фотографий и т. п., в случаях, когда искажения недопустимы или нежелательны. Сжатие с потерями, обладающее значительно большей, чем сжатие без потерь, эффективностью, обычно применяется для сокращения объёма аудио- и видеоданных и цифровых фотографий в тех случаях, когда такое сокращение является приоритетным, а полное соответствие исходных и восстановленных данных не требуется.

В самом общем виде, исходя из принципов реализации, можно выделить организационные, энергетические, сигнальные и пространственные методы защиты от радиопомех.

Организационный метод в простом варианте предполагает такоерасположение источников радиосигналов и такой выбор частот, при которых радиоэлектронные средства проектируемых систем не будут создавать взаимные помехи. Очевидно, что в настоящее время этот метод частотно-территориального разноса в условиях мегаполисов и промышленно развитых регионов, насыщенных радиоэлектронными средствами, становится не слишком эффективным. Однако он утвердился и применяется в форме, требующей выполнения обязательных, определенных международным и национальным регламентами процедур взаимной координации и регистрации полос радиочастот различных сетей. В ходе выполнения указанных процедур координации операторы должны прийти к соглашению о взаимоприемлемых соотношениях сигнал/помеха и тем

самым достичь необходимого уровня электромагнитной совместимости. Для этой цели широко применяется метод поляризационной развязки. При необходимости используется метод частотного сегментирования, который ограничивает используемый частотный ресурс, но операторы вынуждены идти на это для достижения координационных соглашений на взаимоприемлемых условиях.

20–30 лет назад казалось, что строгое выполнение регламентных процедур и соблюдение соглашений обеспечит с высокой степенью вероятности функционирование систем связи без взаимных неприемлемых помех. Однако в настоящее время известные российские специалисты по радиочастотному обеспечению полагают, что в спутниковой связи наступает кризис, связанный именно с самой системой распределения радиочастотного ресурса. Многие спутниковые операторы признают, что современные сети спутниковой связи, прошедшие все этапы координации и регистрации, тем не менее испытывают все больший уровень неприемлемых помех. Это означает, что организационный метод защиты от помех, основанный на действующих регламентных процедурах, во многом исчерпал себя и не может быть признан достаточно эффективным. Несмотря на это, указанный метод распределения частот, основанный на международных и национальных регламентах, является основным инструментом радиочастотного регулирования и сдерживания "радиочастотной анархии".

Энергетический метод борьбы с помехами предусматривает увеличение мощности передатчика до уровня, гарантировано превышающего возможные помехи. Он достаточно широко используется в специальных и военных системах спутниковой связи, однако его применение входит в противоречие с необходимостью обеспечения электромагнитной совместимости, регламентными ограничениями и, кроме того, является энергетически затратным.

Благодаря быстрому развитию цифровой техники в последние 20 лет стало возможным реализовать на практике сигнальные методы помехозащиты, основанные на цифровой обработке сигнала и позволяющие обеспечить снижение воздействия помех на уровне 20…30 дБ. Это, прежде всего, применение псевдослучайных, многочастотных и широкополосных шумоподобных сигналов, а также методов помехоустойчивого кодирования сигнала. Они широко используются в современных системах спутниковой связи и демонстрируют удовлетворительную эффективность. Главный недостаток этих методов – необходимость расширения (в некоторых случаях весьма существенного) радиочастотного спектра для обеспечения защиты от радиопомех. В условиях естественной ограниченности радиочастотного ресурса это существенный недостаток, который снижает эффективность применения таких методов, особенно в высокоскоростных системах. Известно, что применение сигнальных методов приводит к снижению коэффициента помехозащиты пропорционально увеличению скорости потока информации. Несмотря на указанные недостатки, сигнальные методы остаются весьма эффективными, постоянно совершенствуются, и следует ожидать, что они будут востребованы и в перспективе, особенно в сочетании с некоторыми методами пространственной помехозащиты.

Последние разрабатываются и применяются уже не один десяток лет. Наиболее простые из них – экранирование радиоэлектронных средств в направлении воздействия помех и применение радиопоглощающих покрытий вопределенных зонах зеркала антенны для снижения влияния приема помехи по боковым лепесткам диаграммы направленности антенной системы. Эти виды методов заняли свое место, но не получили широкого распространения ввиду того, что они не всегда способны обеспечить необходимый уровень защиты от радиопомех. Например, экранирование не обеспечивает надежной помехозащиты при случайном воздействии помех с неопределенного направления и при этом предполагает создание довольно громоздких конструкций. Радиопоглощающие покрытия имеют ограничения по уровню снижения помех, который далеко не всегда достаточен.

Метод радиоэлектронной компенсации помех, или пространственной режекции помех (наиболее сложный с точки зрения технической реализации), основан на воспроизведении копии мешающего сигнала, подлежащего подавлению. Отметим, что коэффициент помехозащиты этого метода, в отличие от сигнального, практически не зависит от скорости передачи информации. Его эффективность зависит от точности воспроизведения копии сигнала помехи и по некоторым оценкам [2], может достигать уровня подавления помехи до 40 дБ. Такой результат вполне возможно получить в лабораторных условиях. На практике же в системе пространственной режекции помех, работающей в реальных условиях воздействия помех с 2–3 направлений, уже сегодня достигнут уровень 20…25 дБ. Совершенствование электронной компонентной базы, а также применение принципиально нового математического аппарата и функционального программного обеспечения на его основе позволят существенно улучшить полученные результаты.

Дальнейшие параграфы посвящены алгоритмам некоторых сжимающих кодов.

5.5. Код Хаффмана

Начиная с этого параграфа мы познакомимся с некоторыми классическими алгоритмами энтропийного кодирования. Начнем с кодирования Хаффмана.

Дэвид Хаффман придумал этот популярный по сей день алгоритм в 1952 году, когда учился в аспирантуре Массачусетского технологического института. Позже этот американский ученый еще многое сделает для цифровой эпохи, внеся свой вклад в электронику и теорию информации.

Сам алгоритм обладает как несомненными преимуществами, так и некоторыми недостатками. Напомним, что любой энтропийный код имеет основной идеей следующее: редко встречающиеся символы следует кодировать длинными кодовыми словами, а частые – короткими. Очевидно, что построение таки кодов основано на частотном распределении символов кодируемого сообщения, а значит текст должен быть предварительно обработан: посчитаны все символы, количество их упоминаний в сообщении, построена частотная модель сообщения. Это первый недостаток – необходимость двойной обработки сообщения для построения модели в первый раз и непосредственного кодирования – во второй.

Если при кодировании сообщения вы выбираете сжимающий алгоритм Хаффмана, то ваш собеседник не сможет однозначно восстановить информацию, не зная частотного распределения, на основе которого был построен код. Это второй существенный минус данного алгоритма – сжатие может произойти достаточно эффективное, но если текст сообщения был слишком короток, то все усилия на эту экономию будут зачеркнуты передаваемой таблицей с частотным распределением, без которой декодер будет бессилен.

Из несомненных преимуществ этого алгоритма можно выделить префиксность и оптимальность. По построению вы увидите, ни одно кодовое слово не становится началом другого. Оптимальность же заключается в том, что улучшить построенный в результате код уже нельзя, даже перебрав все прочие вариации.

Сам алгоритм основан на построении жадного дерева по модели источника. Жадным оно называется, потому что на каждом этапе выполнения регламентируется выбирать лучший (оптимальный) вариант. Под моделью источника же будем понимать далее частотное распределение кодируемого сообщения. Еще скажем несколько слов о таком понятии как алфавит источника, которое мы будем часто употреблять далее. Под алфавитом источника принято понимать все символы, выдаваемые источником, которые подлежат преобразованию. То есть, это не алфавит языка в нашем естественном понимании, а все без исключения символы: и пробел, и

знаки препинания, и числа, если их нужно кодировать.

Итак, алгоритм Хаффмана принято описывать двумя этапами:

I. Построение дерева
   1. Частотная модель сообщения сортируется по частоте от редких к частым;
   2. Из полученного списка выбирается в нужном количестве (пара, тройка и так далее в зависимости от основания кода) несколько его элементов так, чтобы их суммарная частота была минимальной из всех возможных;
   3. Выбранные элементы заменяются новым – объединенным с суммарной частотой. Шаги 2 и 3 выполняются, пока в списке не останется один элемент, состоящий из всех символов с суммарной частотой 1.

II. Маркировка ветвей
   - Каждое объединение представляет собой несколько ветвей от объединенных элементов к их группировке. На этом этапе в каждом объединении наносятся маркеры (значения, допустимые в коде). Каждая ветка в рамках одного объединения должна быть маркирована уникально.
   - Кодовое слово для символа собирается из маркеров от корня построенного графа по всем веткам, ведущим к листу этого символа.

Предположим, у нас есть некий источник информации, который выдает сигналы A, B, C, D, E с некоторой частотой. Если не учитывать эту частоту, то для пяти сигналов нам понадобится не менее 3 бит для кодирования их блочным кодом, например, можно предложить такой код:

Сигнал	**Кодовое слово**
A	001
B	010
C	011
D	100
E	101

Предположим, что за минуту наш источник выдал следующее:

...ACBEBDEBDEBCDBEABEDBECEBADBEAC...

Примем эту последовательность за среднестатистическое его поведение (за неимением никакой другой информации) и рассчитаем частотное распределение его алфавита.

Сигнал	**Количество за минуту**	**Частота = количество встреч сигнала / общее количество сигналов**
A	4	4/30 = 2/15 = 0,1(3)
B	9	9/30 = 3/10 = 0,3

C	4	4/30 = 2/15 = 0,1(3)
D	5	5/30 = 1/6 = 0,1(6)
E	8	8/30 = 4/15 = 0,2(6)

Здесь для порядка мы привели именно частотное распределение (то есть посчитали частоту), на деле же достаточно использовать второй столбец приведенной таблицы и строить дерево на суммах количеств, а не частот. Из расчёта очевидно, что эти величины прямо пропорциональны, а обращаться к с целыми числами просто удобнее. Поэтому далее дерево будет построено именно на основе количества, а не частоты. Строго говоря, не все энтропийные коды допускают такое в своем построении, но об этом будет сообщено позже в соответствующей главе.

Теперь можно строить энтропийный код. Для начала следует расположить все сигналы в порядке приоритетов. Самый частый сигнал должен стать самым последним, тогда очередь до него дойдет позже, и он получит меньше бит в свое кодовое слово. Самые редкие сигналы должны стать первыми, тогда они обрастут большим количеством бит к концу построения. Таким образом мы добьемся исполнения основной идеи энтропийного кодирования: редким – длинное кодовое слово, частым – короткое. Итак, на первом этапе после сортировки получаем следующее:

Сигнал	**Количество за минуту**
A	4
C	4
D	5
E	8
B	9

И вот мы в первый раз сталкиваемся с неоднозначно трактуемой ситуацией. Существуют два сигнала A и C, имеющие одну частоту. Как с ними поступить? Строго говоря, алгоритм действительно ничего на этот счет не указывает и решение принимает тот, кто организует этот код. Обычно принять согласовывать такие ситуации отдельно. Мы здесь определим эти два сигнала в алфавитном порядке.

На втором этапе нужно строить само дерево постепенно соединяя вершины. Каждый раз нужно выбирать самые «легкие» по весу вершины и объединять их в новую. В каком количестве их брать – говорит основание вашего кода. Будем строить классический бинарный код, а значит нужно объединять по две вершины за раз. Первое объединение коснется двух самых верхних сигналов в нашей таблице: A и C. Теперь вместо них будем рассматривать новую вершину AC с общей суммой 8:

Сигнал	**Количество за минуту**	**Первое объединение**

A	4	AC – 8
C	4	
D	5	D – 5
E	8	E – 8
B	9	B – 9

Теперь в нашем распоряжении стало четыре вершины (AC, D, E, B), и мы вновь должны выбрать две для следующего объединения. Тут нас вновь подстерегает не регламентируемое поведение. Есть два варианта объединения: ACD и DE. Суммарно оба они дадут новую вершину с весом 13. Какое же из них выбрать? Тут вновь нет ответа в общем алгоритме и нужно принимать решение разработчику системы. Главное – принять такое решение один раз на весь вод и не менять его в следующих подобных ситуациях. В качестве возможных решений приведем здесь два:

- Выбирать «по алфавиту»: брать ту пару вершин, что выше в частотном распределении;
- Выбирать по объему: брать ту пару вершин, что будет содержать меньше сигналов.

Для данного примера выберем второй предложенный вариант и будем теперь во всем дереве в подобных ситуациях выбирать пары с меньшим объемом. Тогда после второго соединения образуется новая вершина DE с весом 13:

Сигнал	**Количество за минуту**	**Первое объединение**	**Второе объединение**
A	4	AC – 8	AC – 8
C	4		
D	5	D – 5	DE – 13
E	8	E – 8	
B	9	B – 9	B – 9

Далее продолжаем искать такие пары и объединять их. На третьем шаге лучшим выбором станет пара вершин AC и B с общим весом 17. В данном формате таблицы это неудобно представлять, в конце данного примера будет приведено изображение более классического дерева в виде графа.

Сигнал	**Количество за минуту**	**Первое объединение**	**Второе объединение**	**Третье объединение**
A	4	AC – 8	AC – 8	ACB – 17
C	4			

D	5	D – 5	DE – 13	DE – 13
E	8	E – 8		
B	9	B – 9	B – 9	

Финальное объединение уже окончательно соберет все сигналы в корень дерева. Всегда проверьте, что вес корня совпал с общим количеством сигналов, которое вы получили в начальной строке (если же вы работаете с частотами, то вес корня должен быть равен 1).

Сигнал	**Количество за минуту**	**Первое объединение**	**Второе объединение**	**Третье объединение**	**Итог**
A	4	AC – 8	AC – 8	ACB – 17	ACBDE – 30
C	4				
D	5	D – 5	DE – 13	DE – 13	
E	8	E – 8			
B	9	B – 9	B – 9		

Теперь, когда мы закончили с построением дерева, нужно приступать к третьему этапу – маркировке ветвей. И в последний раз нам нужно прийти к одной договоренности, которая строго не регламентируется алгоритмом. Маркировать можно верхнюю ветку нулем, а нижнюю – единицей, и наоборот. Решение вновь должен принять разработчик и не менять его до конца построения кода. Мы будем маркировать верхнюю ветку нулем. Тогда получим следующее: в нашей построенной таблице будут стерты промежуточные объединения, а сами ячейки и будут выполнять роль двух ветвей.

Сигнал	**Количество за минуту**	**Первое объединение**	**Второе объединение**	**Третье объединение**	**Итог**
A	0	0		0	корень
C	1				
D	0		1		
E	1				
B	1				

Осталось собрать кодовые слова, шествуя от корня к сигналу и собирая по пути все маркеры.

Сигнал	Кодовое слово
A	`000`
C	`001`
D	`10`
E	`11`
B	`01`

Как видите, в сравнении с изначальным блочным кодом мы сэкономили хотя бы три бита. На самом деле экономия еще больше, но сравнивать построенные коды нужно не посимвольно, а в общем. Для этого прибегнем к такому математическому инструменту, как математическое ожидание, которое покажет нам среднюю длину кодового слова построенного кода Хаффмана. Это будет не среднее арифметическое, а среднее с учетом частоты. Формула для расчета математического ожидания такова:

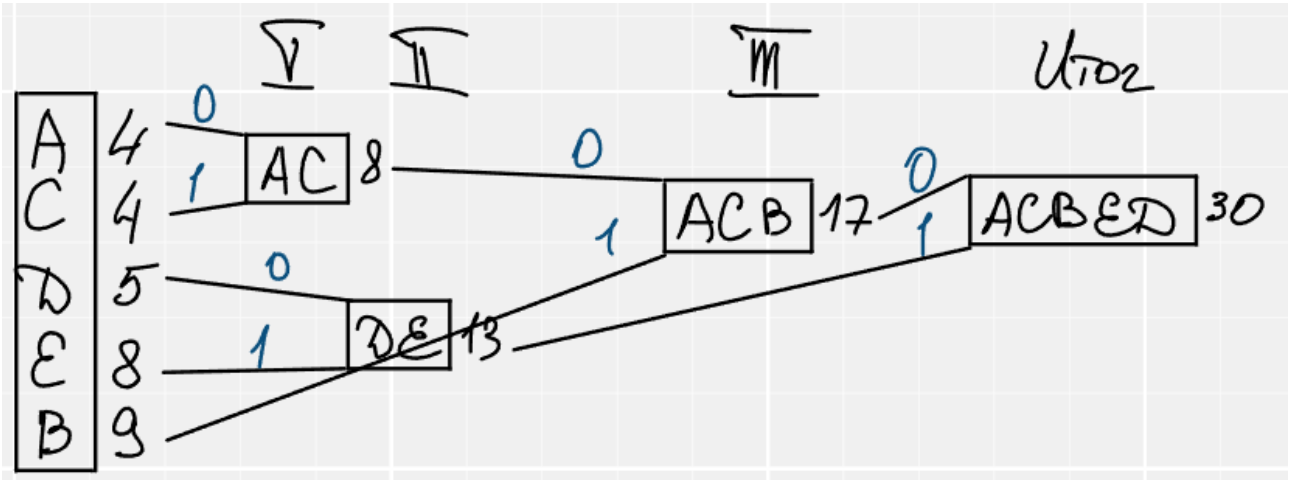
$$M = \sum_{i=1}^{n} p_i \cdot l_i,$$

где n – количество символов в алфавите источника, p_i – частота i – го символа, l_i – длина кодового слова i – го символа. В нашем случае получим:

Сигнал	Количество за минуту	Длина кодового слова
A	4	3
C	4	3
D	5	2
E	8	2
B	9	2

$$M = \frac{1}{30}(4 \cdot 3 + 4 \cdot 3 + 5 \cdot 2 + 8 \cdot 2 + 9 \cdot 2) = \frac{68}{30} = \frac{34}{15} = 2{,}2(6)$$

При примерном округлении до тысячных средняя длина кодового слова нашего кода Хаффмана оказывается равной 2,267, а значит в сравнении с блочным кодом нам удалось выиграть 0,733 бита на каждый сигнал! Вот это экономия. Напоследок приведем ниже более классическое представление все того же дерева (после маркировки), чем рассмотренная ранее табличная форма.



Обратите внимание, что построенный код получился префиксным (но не постфиксным). И действительно по построению код Хаффмана обладает таким замечательным свойством. Подумайте, что нужно изменить в алгоритме, чтобы он стал постфиксным?

Рассмотрим еще один пример, на этот раз с текстовым сообщением и троичной системой кодирования. Пусть нужно закодировать такой текст:

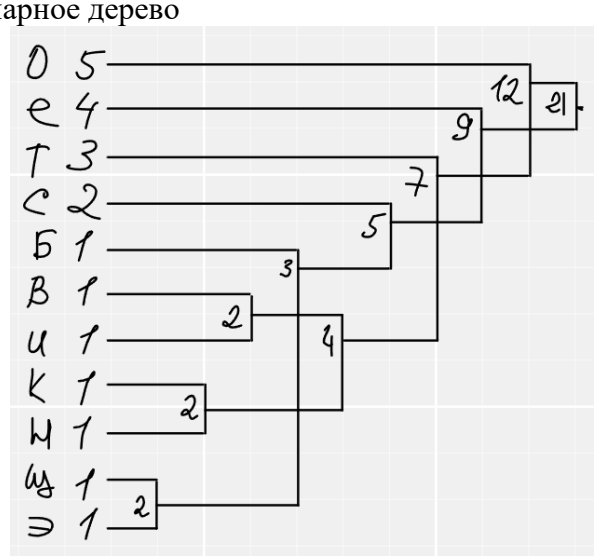
«Это текстовое сообщение»

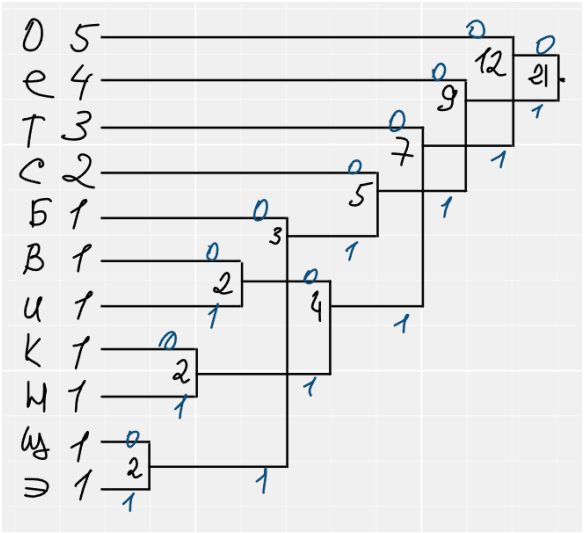
Построим частотное распределение, приняв данный текст за среднестатистическое сообщение некоторого информационного источника. Ниже приведено уже отсортированное распределение (по количеству).

Символ	**Количество в тексте**
О	5
Е	4
Т	3
С	2
Б	1
В	1
И	1
К	1
Н	1
Щ	1
Э	1

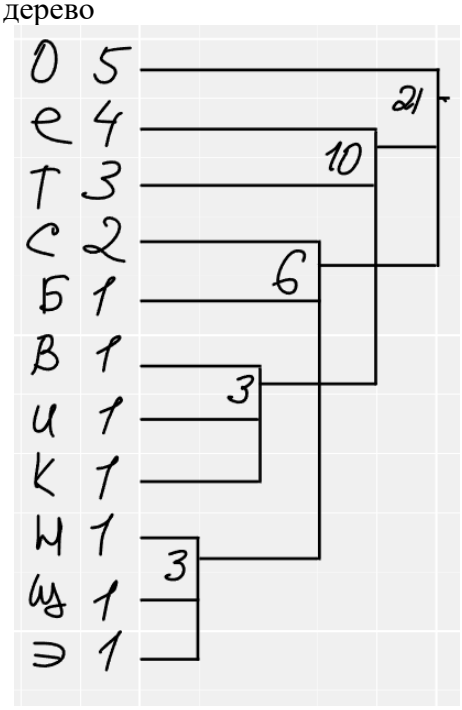
Далее опустим подробное описание построения дерева, приведем лишь итог. Читателю предлагается самостоятельно проанализировать все ходы и решения, а лучше попробовать построить подобное дерево самим (самостоятельно предположив все разумные допущения в не регламентируемых алгоритмом ситуациях), а затем сверить с нашим результатом. Построение производилось для бинарного кода с алфавитом 0 и 1 и тернарного (с алфавитом +, 0 и –).

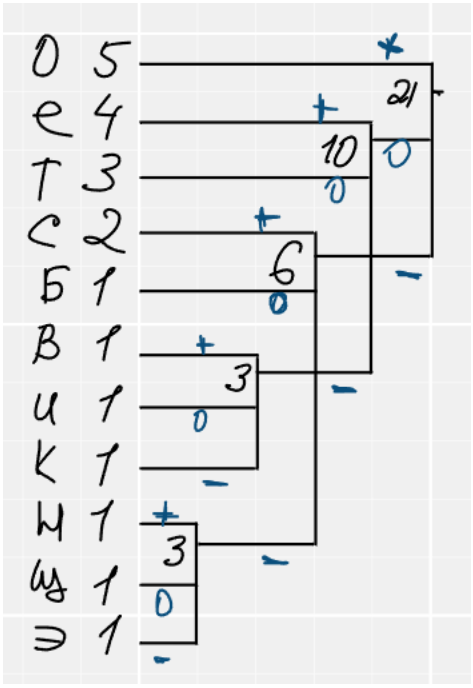
Бинарное дерево





Тернарное дерево





В результате получаем следующие коды. Для визуализации оптимальности подхода приводим также простейшие блочные коды в сравнении.

Символ	Количество в тексте	Простейший блочный двоичный	Бинарное дерево Хаффмана	Простейший блочный троичный	Тернарное дерево Хаффмана

О	5	0001	00	+++	+
Е	4	0010	10	++0	0+
Т	3	0011	010	++-	00
С	2	0100	110	+0+	-+
Б	1	0101	1110	+00	-0
В	1	0110	01100	+0-	0-+
И	1	0111	01101	+-+	0-0
К	1	1000	01110	+-0	0--
Н	1	1001	01111	+--	--+
Щ	1	1010	11110	0++	--0
Э	1	1011	11111	0+0	---
Средняя длина		**4**	**3,190**	**3**	**2,048**

Напоследок, рассчитаем энтропию нашего сообщения для вычисления избыточности. Напомним формулу информационной энтропии (двоичной H_2 и троичной H_3):

$$H_2 = -\sum_{1 \le i \le N} p_i \cdot \log_2 p_i\,, \qquad H_3 = -\sum_{1 \le i \le N} p_i \cdot \log_3 p_i$$

Под избыточностью же понимают в таком случае разность средней длины кода и энтропии сообщения, то есть $\nu = M - H$.

Итак, двоичная энтропия нашего сообщения составляет

$$H_2 = -\left(\frac{5}{21}\cdot\log_2\frac{5}{21} + \frac{4}{21}\cdot\log_2\frac{4}{21} + \frac{3}{21}\cdot\log_2\frac{3}{21} + \frac{2}{21}\cdot\log_2\frac{2}{21} + \right.$$
$$\left. +7\cdot\frac{1}{21}\cdot\log_2\frac{1}{21}\right) \approx 3{,}137$$

Тогда избыточность на символ сообщения для двоичного кода составит примерно $\nu_{\text{Хафф}} = M_2 - H_2 = 3{,}190 - 3{,}137 = 0{,}053$ (бит/символ), что в сравнении с избыточностью простого блочного кода ($\nu_{\text{блоч}} = 4 - 3{,}137 = 0{,}863$ бит/символ) лучше в 16 раз.

Для построенного троичного кода результаты таковы.

$$H_3 = -\left(\frac{5}{21}\cdot\log_3\frac{5}{21} + \frac{4}{21}\cdot\log_3\frac{4}{21} + \frac{3}{21}\cdot\log_3\frac{3}{21} + \frac{2}{21}\cdot\log_3\frac{2}{21} + \right.$$
$$\left. +7\cdot\frac{1}{21}\cdot\log_3\frac{1}{21}\right) \approx 1{,}979$$

Тогда избыточность на символ сообщения для троичного кода составит примерно $\nu = M_3 - H_3 = 2{,}048 - 1{,}979 = 0{,}069$ (бит/символ), что в сравнении с избыточностью простого блочного кода ($\nu_{\text{блоч}} = 3 - 1{,}979 = 1{,}021$ бит/символ) лучше почти в 15 раз.

5.6. Код Шеннона-Фано

В этом параграфе рассмотрим другой алгоритм энтропийного кодирования – код Шеннона-Фано. Этот сжимающий код был придуман одним из первых сразу двумя американскими учеными, работавшими над своими трудами отдельно.

Клод Элвуд Шеннон считается отцом-основателем теории информации в целом, и многое сделал для появления такой науки, как криптографии. Именно он придумал понятие «бит», которое знакомо на сегодняшний день любому современному человеку. Мы уже не раз упоминали этого выдающегося ученного в данной книге, например, говоря об информационной энтропии.

Роберт Марио Фано (также уже известный нам благодаря своим условиям префиксности кодов) был известным итало-американским ученым, внесшим свой вклад в теорию информации.

К характерным особенностям данного алгоритма можно отнести префиксность, которой он обладает также по построению. Однако в оптимальности данный код может проиграть алгоритму Хаффмана. Это происходит из-за неоднозначно регламентируемых шагов, которые также присутствовали и в предыдущем алгоритме, однако код Хаффмана дает устойчиво оптимальный результат при любом выборе разработчика, в отличие от алгоритма Шеннона-Фано, более чувствительного к такому выбору. В этом алгоритме выбор оптимального разбиения на каждом шаге не гарантирует оптимального результата в конце. Однако, если построить все возможные коды Шеннона-Фано на конкретной модели источника, среди них обязательно будут и все коды Хаффмана, построенные на той же модели.

Визуально алгоритм Шеннона-Фано очень похож на алгоритм Хаффмана. Будет построено такое же дерево, но в обратном порядке. Теперь мы будем двигаться от корня к листьям. Как и любой энтропийный код, данный алгоритм начинается с частотного распределения алфавита источника. Затем вместо объединения листьев до корня мы будем делать обратное действие – поэтапно разбивать множество на подмножества до тех пор, пока в каждом подмножестве не останется по одному элементу (листу графа). Далее опять следует привычная нам маркировка. Опишем алгоритм:

I. Построение дерева
    1. Частотная модель сообщения сортируется по частоте от редких к частым. Назовем список частот множеством;
    2. Полученное множество нужно разделить на несколько подмножеств (количество вновь исходит от основания кода, который мы должны построить) так, чтобы сумма частот в каждом из них была примерно равна, при этом не допускается перемешивание. Более формально из всех разбиений следует выбрать то, для которого разность сумм частот подмножеств меньше всего;
    3. Полученные подмножества по отдельности вновь подвергаются шагу 2 до тех пор, пока в подмножестве не останется один элемент.

II. Маркировка ветвей
    – На этом этапе каждому подмножеству присваивается маркер (значение, допустимое в коде). Каждое подмножество в рамках одного множества должен быть маркирован уникально.
    – Кодовое слово для символа собирается из маркеров от корня построенного графа по всем подмножествам, ведущим к листу этого символа.

Из описанного алгоритма становится очевидно, где нас поджидает неоднозначно регламентируемое поведение: то делать, если есть два разбиения с минимальной равной разностью суммарных вероятностей подмножеств? Тут решение исходит опять от разработчика, но оно должно быть принято единожды и не меняться во время всего построения.

Вновь построим кодовую таблицу для некоторого источника из предыдущего параграфа. Итак, модель источника такова (приведена в сортированном виде):

Сигнал	**Количество за минуту**	**Частота = количество встреч сигнала / общее количество сигналов**
A	4	4/30 = 2/15 = 0,1(3)
C	4	4/30 = 2/15 = 0,1(3)
D	5	5/30 = 1/6 = 0,1(6)
E	8	8/30 = 4/15 = 0,2(6)
B	9	9/30 = 3/10 = 0,3

Этот код также позволяет работать не с частотами, а с количеством использования сигнала в единицу времени. Этим и воспользуемся, чтобы работать с целыми числами.

Для построения бинарного кода нам необходимо выполнять разбиение на два подмножества. Рассмотрим возможные варианты (напомним, что перемешивать сигналы нельзя):

Вариант 1. Отделить A (суммарный вес 4) и CDEB (суммарный вес 26). Разность такого варианта составляет 22.

Вариант 2. Отделить AC (суммарный вес 8) и DEB (суммарный вес 22). Разность такого варианта составляет 14.

Вариант 3. Отделить ACD (суммарный вес 13) и EB (суммарный вес 17). Разность такого варианта составляет 4.

Вариант 4. Отделить ACDE (суммарный вес 21) и B (суммарный вес 9). Разность такого варианта составляет 12.

Минимальную разницу имеет вариант 3, значит именно это разбиение мы и предпочтем для первого шага.

Сигнал	**Количество за минуту**	**Первое разбиение**
A	4	
C	4	
D	5	
E	8	
B	9	

Теперь рассмотрим полученные подмножества отдельно таким же образом. Для множества ACD есть два варианта разбиения:

Вариант 1. Отделить A (суммарный вес 4) и CD (суммарный вес 9). Разность такого варианта составляет 5.

Вариант 2. Отделить AC (суммарный вес 8) и D (суммарный вес 5). Разность такого варианта составляет 3.

Предпочтем второй вариант.

Для множества EB нескольких вариантов нет, он единственный. Далее с этим множеством и его подмножествами работу вести не станем, так как в результате получаем подмножества из одного элемента.

Сигнал	**Количество за минуту**	**Первое разбиение**	**Второе разбиение**
A	4		
C	4		
D	5		
E	8		
B	9		

Последнее разбиение для множества AC очевидно – есть лишь один вариант, к нему и прибегнем. В результате получаем окончательно построенный граф, готовый к маркировке

Сигнал	**Количество за минуту**	**Первое разбиение**	**Второе разбиение**	**Третье разбиение**
A	4			
C	4			
D	5			
E	8			
B	9			

Теперь каждое разбиение получает свой маркер. Для порядка положим, что верхнее подмножество будет маркироваться нулем, а нижнее – единицей.

Сигнал	**Количество за минуту**	**Первое разбиение**	**Второе разбиение**	**Третье разбиение**
A	4	0	0	0
C	4			1

D	5		1
E	8	1	0
B	9		1

В результате собранные от корня кодовые слова для этих сигналов получаются таковыми:

Сигнал	**Кодовое слово**
A	`000`
C	`001`
D	`01`
E	`10`
B	`11`

Очевидно, что показатели средней длины и избыточности данного кода совпадут с результатами для кода Хаффмана, построенного на этой же модели. Так как хоть кодовые слова и получились разными, их длины совпадают.

Также проведем кодирование по алгоритму Шеннона-Фано и для другого примера предыдущего параграфа:

«Это текстовое сообщение»

Начнем с двоичного кода. Приведем ниже частотное распределение.

Символ	**Количество в тексте**
О	5
Е	4
Т	3
С	2
Б	1
В	1
И	1

К	1
Н	1
Щ	1
Э	1

Первое разбиение выбирается из вариантов (не будем приводить далее все возможные, укажем только те, что близки к половине суммарного веса, то есть 10):

Вариант 1. Отделить ОЕ (суммарный вес 9) и ТС…Э (суммарный вес 12). Разность такого варианта составляет 3.

Вариант 2. Отделить ОЕТ (суммарный вес 12) и СБ…Э (суммарный вес 9). Разность такого варианта составляет 3.

Уже на первом же шаге мы встречаем неоднозначный выбор, так как суммарный вес у всех символов – число нечетное (21), то разбить его на два множества можно двумя равнозначными способами. Тут решение остается за разработчиками, и мы предпочтем такой выбор: из двух равнозначных разбиений оставляем то, в котором подмножества оказываются ближе по объему, если этот признак также не дает однозначного выбора, оставляем то разбиение, при котором вес «верхнего» (первого) подмножества больше. В нашем случае модно однозначно переделиться по первому правилу – мощности подмножеств ближе во втором варианте, его и предпочтем. Действительно, в первом варианте мощность ОЕ – 2, мощность ТС…Э – 9 (разница – 7). Во втором варианте мощность ОЕТ – 3, мощность СБ…Э – 8(разница – 5).

Итак, получаем следующее

Символ	**Количество в тексте**	**Первое разбиение**
О	5	
Е	4	
Т	3	
С	2	
Б	1	
В	1	
И	1	
К	1	
Н	1	
Щ	1	

Э	1	

Второе разбиение верхнего подмножества ОЕТ в комментариях не нуждается, а вот нижнее подмножество СБ…Э дает нам опять неоднозначный выбор:

Вариант 1. Отделить СБВ (суммарный вес 4) и ИК…Э (суммарный вес 5). Разность такого варианта составляет 1.

Вариант 2. Отделить СБВИ (суммарный вес 5) и КНЩЭ (суммарный вес 4). Разность такого варианта составляет 1.

Вновь по установленному нами ранее правилу анализируем объемы подмножеств этих двух разбиений:

Вариант 1. Мощность СБВ – 3, а ИК…Э – 5. Разность такого варианта составляет 2.

Вариант 2. Мощность СБВИ – 4, а КНЩЭ – 4. Разность такого варианта составляет 0.

Определяем лучшим второй вариант разбиения. Дальнейшие шаги останутся без подробных комментариев. Читателю рекомендуется самостоятельно проанализировать их.

Символ	**Количество в тексте**	**Первое разбиение**	**Второе разбиение**	**Третье разбиение**	**Четвертое разбиение**
О	5	0	0		
Е	4		1	0	
Т	3			1	
С	2	1	0	0	0
Б	1				1
В	1			1	0
И	1				1
К	1		1	0	0
Н	1				1
Щ	1			1	0
Э	1				1

Теперь расставим маркеры. Вновь положим в верхнее подмножество каждый раз ставить 0, а в нижнее – 1. В результате получили код, представленный в сравнительной таблице ниже.

Рассмотрим ту же модель, но допустим при равнозначных разбиениях всегда выбирать то, что делает первое подмножество меньшим по весу. Опустим комментарии по построению и приведем ниже лишь маркированный итоговый граф:

Символ	Количество в тексте	Первое разбиение	Второе разбиение	Третье разбиение	Четвертое разбиение	Пятое разбиение
О	5	0	0			
Е	4		1			
Т	3	1	0	0		
С	2			1	0	
Б	1				1	
В	1		1	0	0	
И	1				1	0
К	1					1
Н	1			1	0	
Щ	1				1	0
Э	1					1

Теперь приведем тут полученные коды и сравнить показатели (для сравнения в таблице представлены и результаты предыдущего параграфа):

Символ	Количество в тексте	Простейший блочный двоичный	Бинарное дерево Хаффмана	Код Шеннона-Фано	Код Шеннона-Фано (2)
О	5	0001	00	00	00
Е	4	0010	10	010	01
Т	3	0011	010	011	100
С	2	0100	110	1000	1010
Б	1	0101	1110	1001	1011
В	1	0110	01100	1010	1100
И	1	0111	01101	1011	11010
К	1	1000	01110	1100	11011
Н	1	1001	01111	1101	1110

Щ	1	1010	11110	1110	11110
Э	1	1011	11111	1111	11111
Средняя длина		**4**	**3,190**	**3,190**	**3,190**
Избыточность		**0,863**	**0,054**	**0,054**	**0,054**

Занимательно, что несмотря на разные кодовые слова, показатели избыточности на символ оказались одинаковыми.

Однако заметим, что, избрав другое поведение при не регламентируемом алгоритмом Шеннона-Фано поведении, можно получить другой код с худшими показателями.

Напоследок приведем тут результаты для кодирования алгоритмом Шеннона-Фано того же сообщения троичным кодом. Читателю предлагается самостоятельно проанализировать выбранные допущения в нашем алгоритме. А лучше всего перед этим самостоятельно попрактиковать построение этого кода, сверив результаты в конце.

Символ	**Количество в тексте**	**Первое разбиение**	**Второе разбиение**	**Третье разбиение**
О	5	+	+	
Е	4		0	
Т	3	0	+	
С	2		0	
Б	1		-	
В	1	-	+	+
И	1			0
К	1		0	+
Н	1			0
Щ	1		-	+
Э	1			0

Результаты:

Символ	Количество в тексте	Простейший блочный троичный	Тернарное дерево Хаффмана	Код Шеннона-Фано
О	5	+++	+	++
Е	4	++0	0+	+0
Т	3	++-	00	0+
С	2	+0+	-+	00
Б	1	+00	-0	0-
В	1	+0-	0-+	-++
И	1	+-+	0-0	-+0
К	1	+-0	0--	-0+
Н	1	+--	--+	-00
Щ	1	0++	--0	--+
Э	1	0+0	---	--0
Средняя длина		**3**	**2,048**	**2,286**
Избыточность		**1,021**	**0,068**	**0,307**

В данном случае очевидно, что построенный код Шеннона-Фано проигрывает коду Хаффмана по показателю избыточности на символ в 4,5 раза, хотя все еще показывает себя в почти 3, раза лучше простейшего блочного кода.

5.7. Код Шеннона

Последний энтропийный код, с которым мы познакомимся, был придуман Шенноном в 1948 году для доказательства своей теоремы о помехоустойчивом кодировании. Этот код принято считать первым энтропийным кодом.

Хотя по оптимальности он отстает от предыдущих рассмотренных нами алгоритмов, его несомненный плюс – простота построения. Кодовые слова строятся по-прежнему по отсортированной модели источника (так как код является энтропийным), но вместо громоздких в программной реализации алгоритмов разбиения или объединения множеств, тут вычисления строятся всего на двух формулах: для расчета длины кодового слова и для расчета его значения.

Итак, опишем алгоритм кода Шеннона:

I. Расчет длины кодового слова

– Для определения длины i-го кодового слова прибегнем к формуле от частоты это слова (p_i):

$$l_i = \lceil -\log_2 p_i \rceil$$

Здесь используется операция округления вверх $\lceil a \rceil = \min\{b: b - 1 < a \leq b\}$. Например, $\lceil 3{,}2 \rceil = 4$.

II. Расчет i-го кодового слова:

1. Сначала для i-го символа вычисляется накопленная сумма частот q_i по формуле:

$$q_i = \sum_{j=1}^{i-1} p_j$$

При этом для первого символа накопленная сумма полагается равной 0: $q_1 = 0, q_2 = p_1, q_3 = p_1 + p_2, q_4 = p_1 + p_2 + p_3, \ldots$

2. Затем эта сумма переводится в двоичную систему счисления.
3. В полученном двоичном числе нужно взять после запятой столько символов, сколько показала длинна l_i.

Построение кода удобно представлять таблицей. Проведем указанные манипуляции для примера уже известного нам источника с пятью сигналами. На этот раз алгоритм жестко требует от нас работы с частотами.

Сигнал	**Частота**	**Длина**	**Накопленная сумма**	**Двоичное представление**	**Кодовое слово**
B	0,3000	2	0,0000	0,00000…	`00`
E	0,2667	2	0,3000	0,01001…	`01`
D	0,1667	3	0,5667	0,10010…	`100`
A	0,1333	3	0,7334	0,10111…	`101`
C	0,1333	3	0,8667	0,11011…	`110`

Сравним результаты с полученными ранее кодами. Заметим, что энтропия рассматриваемой системы равна $H = 2{,}236$.

Сигнал	**Простейший блочный код**	**Код Хаффмана**	**Код Шеннона-Фано**	**Код Шеннона**
A	`001`	`000`	`000`	`101`
C	`010`	`001`	`001`	`110`
D	`011`	`10`	`01`	`100`
E	`100`	`11`	`10`	`01`
B	`101`	`01`	`11`	`00`

Средняя длина	**3**	**2,267**	**2,267**	**2,433**
Избыточность	**0,764**	**0,031**	**0,031**	**0,198**

В результате, как и предсказывалось код Шеннона не стал самым оптимальным, проиграв кодам Хаффмана и Шеннона-Фано почти в 6,5 раз. Но он лучше блочного (в силу своей энтропийности) почти в 4 раза.

Заметим также, что полученный код можно оптимизировать, то есть проанализировать полученные кодовые слова и самостоятельно изменить их с сохранением префиксности.

Например, кодовое слово сигнала D исправить не выйдет, так как возможные варианты испортят однозначное декодирование кода:

- Если изменить кодовое слово D = 100 на 10 (убрав последний бит), то будет конфликт с кодовым словом A = 101 в префиксности или и кодовым словом C = 110 в постфиксности;
- Если изменить кодовое слово D = 100 на 00 (убрав первый бит), то будет точное совпадение с кодовым словом B = 00.

Однако можно улучшить кодовое слово C = 110 также двумя способами:

- Можно убрать последний бит и заменить кодовое слово C = 110 на 11. Тогда сохранится префиксность кода;
- Можно убрать первый бит и заменить кодовое слово C = 110 на 10. Тогда хотя и потеряно будет свойство префиксности (в силу конфликта с кодовым словом A = 101) однако будет сохранено свойство постфиксности, а для однозначного восстановления этого достаточно.

Оптимизировать код можно и за счет других кодовых слов. Эта процедура требует большого внимания и аналитических способностей.

Процесс оптимизации сильно усложняет алгоритм и сводит на нет основное преимущество данного кода – простоту реализации по двум формулам без анализа. Тем более, как несложно видеть, оптимизированный код по-прежнему проигрывает прочим энтропийным кодам в избыточности на символ в два раза, хотя и улучшил показатель по сравнению с неоптимизированным вариантом в 3 раза.

Сигнал	**Простейший блочный код**	**Код Хаффмана**	**Код Шеннона-Фано**	**Код Шеннона**	**Оптимизированный код Шеннона**
A	001	000	000	101	101
C	010	001	001	110	11
D	011	10	01	100	100
E	100	11	10	01	01
B	101	01	11	00	00
Средняя длина	**3**	**2,267**	**2,267**	**2,433**	**2,3**

Избыточ-ность	**0,764**	**0,031**	**0,031**	**0,198**	**0,064**

Этот метод не дает оптимального результата и итоговые кодовые слова можно оптимизировать, однако не сильно. Несомненным преимуществом данного метода кодирования является алгоритмичность – для вычисления кодового слова требуется лишь знать частоту в двоичном представлении и посчитать длину по формуле, что безусловно, проще реализовать чем разбиение (или группировку) множеств.

Приведем также таблицы построения кода Шеннона для другого примера предыдущих параграфов.

Сигнал	**Частота**	**Длина**	**Накопленная сумма**	**Двоичное представление**	**Кодовое слово**
О	0,2381	3	0,0000	0,000000…	`000`
Е	0,1905	3	0,2381	0,001111…	`001`
Т	0,1429	3	0,4286	0,011011…	`011`
С	0,0952	4	0,5714	0,100100…	`1001`
Б	0,0476	5	0,6667	0,101010…	`10101`
В	0,0476	5	0,7143	0,101101…	`10110`
И	0,0476	5	0,7619	0,110000…	`11000`
К	0,0476	5	0,8095	0,110011…	`11001`
Н	0,0476	5	0,8571	0,110110…	`11011`
Щ	0,0476	5	0,9048	0,111010…	`11101`
Э	0,0476	5	0,9524	0,111101…	`11110`

Сравним результат с предыдущими кодами.

Символ	**Количество в тексте**	**Простейший блочный двоичный**	**Бинарное дерево Хаффмана**	**Код Шеннона-Фано**	**Код Шеннона**
О	5	`0001`	`00`	`00`	`000`
Е	4	`0010`	`10`	`010`	`001`
Т	3	`0011`	`010`	`011`	`011`

С	2	0100	110	1000	1001
Б	1	0101	1110	1001	10101
В	1	0110	01100	1010	10110
И	1	0111	01101	1011	11000
К	1	1000	01110	1100	11001
Н	1	1001	01111	1101	11011
Щ	1	1010	11110	1110	11101
Э	1	1011	11111	1111	11110
Средняя длина		**4**	**3,190**	**3,190**	**3,762**
Избыточность		**0,863**	**0,054**	**0,054**	**0,625**

Вновь алгоритм Шеннона подтверждает, что энтропийный код лучше блочного (хотя бы почти в полтора раза, как в данном примере), хотя не лучше прочих энтропийных кодов (проигрывая им в 11,5 раз). Оптимизацию данного примера приводить не станем, оставив это задание пытливым читателям.

Также приведем и троичное кодирование. В алгоритме изменится лишь две вещи:

- теперь для расчета длины кодового слова в формуле нужно блать логарифм по основанию 3;
- перевод накопленной суммы следует производить в троичную систему счисления.

Сигнал	**Частота**	**Длина**	**Накопленная сумма**	**Троичное представление**	**Кодовое слово**
О	0,2381	2	0,0000	0,00000…	++
Е	0,1905	2	0,2381	0,02010…	+-
Т	0,1429	2	0,4286	0,10212…	0-
С	0,0952	3	0,5714	0,12010…	0-+
Б	0,0476	3	0,6667	0,20000…	-++
В	0,0476	3	0,7143	0,20102…	-+0
И	0,0476	3	0,7619	0,20212…	-+-

К	0,0476	3	0,8095	0,21021…	-0+
Н	0,0476	3	0,8571	0,21201…	-0-
Щ	0,0476	3	0,9048	0,22010…	--+
Э	0,0476	3	0,9524	0,22120…	--0

Сравним полученный неоптимизированный код с предыдущими:

Символ	**Количество в тексте**	**Простейший блочный троичный**	**Тернарное дерево Хаффмана**	**Код Шеннона-Фано**	**Код Шеннона**
О	5	+++	+	++	++
Е	4	++0	0+	+0	+-
Т	3	++-	00	0+	0-
С	2	+0+	-+	00	0-+
Б	1	+00	-0	0-	-++
В	1	+0-	0-+	-++	-+0
И	1	+-+	0-0	-+0	-+-
К	1	+-0	0--	-0+	-0+
Н	1	+--	--+	-00	-0-
Щ	1	0++	--0	--+	--+
Э	1	0+0	---	--0	--0
Средняя длина		**3**	**2,048**	**2,286**	**2,429**
Избыточность		**1,021**	**0,068**	**0,307**	**0,449**

И вновь код Шеннона оказывается лучше простого блочного (почти в 2,5 раза), но хуже других энтропийных кодов (почти в полтора раза хуже кода Шеннона-Фано, и в 6,6 раз хуже кода Хаффмана).

Раздел 6. Кодирование канала

6.1. Канальное кодирование

В каналах с помехами эффективным средством повышения достоверности передачи сообщений является помехоустойчивое кодирование. Оно основано на применении специальных кодов, которые корректируют ошибки, вызванные действием помех. Код называется корректирующим, если он позволяет обнаруживать или обнаруживать и исправлять ошибки при приеме сообщений. Код, посредством которого только обнаруживаются ошибки, носит название обнаруживающего кода. Исправление ошибки при таком кодировании обычно производится путем повторения искаженных сообщений. Запрос о повторении передается по каналу обратной связи. Код, исправляющий обнаруженные ошибки, называется исправляющим, кодом. В этом случае фиксируется не только сам факт наличия ошибок, но и устанавливается, какие кодовые символы приняты ошибочно, что позволяет их исправить без повторной передачи. Известны также коды, в которых исправляется только часть обнаруженных ошибок, а остальные ошибочные комбинации передаются повторно.

Для того чтобы код обладал корректирующими способностями, в кодовой последовательности должны содержаться дополнительные (избыточные) символы, предназначенные для корректирования ошибок. Чем больше избыточность кода, тем выше его корректирующая способность. Помехоустойчивые коды могут быть построены с любым основанием. Ниже рассматриваются только двоичные коды, теория которых разработана наиболее полно. В настоящее время известно большое количество корректирующих кодов, отличающихся как принципами построения, так и основными характеристиками.

Все известные в настоящее время коды могут быть разделены на две большие группы: блочные и непрерывные. Блочные коды характеризуются тем, что последовательность передаваемых символов разделена на блоки операции кодирования и декодирования в каждом блоке производятся отдельно. Отличительной особенностью непрерывных кодов является то, что первичная последовательность символов, несущих информацию, непрерывно преобразуется по определенному закону в другую последовательность, содержащую избыточное число символов.

Обнаружение ошибок в технике связи – действие, направленное на контроль целостности данных при записи/воспроизведении информации или при её передаче по линиям связи. Исправление ошибок (коррекция ошибок) – процедура восстановления информации после чтения её из устройства хранения или канала связи. В системах связи возможны несколько стратегий борьбы с ошибками:

- обнаружение ошибок в блоках данных и автоматический запрос повторной передачи повреждённых блоков — этот подход применяется, в основном, на канальном и транспортном уровнях;
- обнаружение ошибок в блоках данных и отбрасывание повреждённых блоков – такой подход иногда применяется в системах потокового мультимедиа, где важна задержка передачи и нет времени на повторную передачу;
- исправление ошибок применяется на физическом уровне.

В теории помехоустойчивого кодирования важным является вопрос об использовании избыточности для корректирования возникающих при передаче ошибок. Здесь удобно рассмотреть блочные моды, в которых всегда имеется возможность выделить отдельные кодовые комбинации. Напомним, что для равномерных кодов, которые в дальнейшем только и будут изучаться, число возможных комбинаций равно $M=2^n$, где n — значность кода. В обычном некорректирующем коде без избыточности, например в коде Бодо, число комбинаций M выбирается равным числу сообщений алфавита источника M0 и все комбинации используются для передачи информации. Корректирующие коды строятся так, чтобы число комбинаций M превышало число

сообщений источника М0. Однако в.этом случае лишь М0 комбинаций из общего числа используется для передачи информации. Эти комбинации называются разрешенными, а остальные М—М0 комбинаций носят название запрещенных. На приемном конце в декодирующем устройстве известно, какие комбинации являются разрешенными и какие запрещенными. Поэтому если переданная разрешенная комбинация в результате ошибки преобразуется в некоторую запрещенную комбинацию, то такая ошибка будет обнаружена, а при определенных условиях исправлена. Естественно, что ошибки, приводящие к образованию другой разрешенной комбинации, не обнаруживаются.

6.2. Корректирующая способность

Расстоянием Хемминга (метрикой Хемминга) между двумя кодовыми словами $\vec{u}$ и $\vec{v}$ называется количество отличных бит на соответствующих позициях: $d_H(\vec{u},\vec{v}) = \sum_s \left|u^{(s)} - v^{(s)}\right|$.

Минимальное расстояние Хемминга $d_{min} = \min_{u \neq v}(\vec{u},\vec{v})$ является важной характеристикой линейного блокового кода. Она показывает, насколько «далеко» расположены коды друг от друга. Она определяет другую, не менее важную характеристику – корректирующую способность: $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$.

Корректирующая способность определяет, сколько ошибок передачи кода можно гарантированно исправить. То есть вокруг каждого кодового слова А имеем t-окрестность A_t, которая состоит из всех возможных вариантов передачи кодового слова А с числом ошибок не более t. Никакие две окрестности двух любых кодовых слов не пересекаются друг с другом, так как расстояние между кодовыми словами (то есть центрами этих окрестностей) всегда больше двух их радиусов $d_H(A,B) \geq d_{min} > 2t$.

Таким образом, получив искажённую кодовую комбинацию из A_t, декодер принимает решение, что исходной была кодовая комбинация А, исправляя тем самым не более t ошибок.

Поясним на примере. Предположим, что есть два кодовых слова А и В, расстояние Хемминга между ними равно 3. Если было передано слово А, и канал внёс ошибку в одном бите, она может быть исправлена, так как даже в этом случае принятое слово ближе к кодовому слову А, чем к любому другому, и, в частности, к В. Но если каналом были внесены ошибки в двух битах (в которых А отличалось от В), то результат ошибочной передачи А окажется ближе к В, и декодер примет решение, что передавалось слово В.

При искажении меньшего числа символов комбинация В перейдет в запрещенную комбинацию, и ошибка будет обнаружена. Отсюда следует, что ошибка всегда обнаруживается, если ее кратность, т. е. число искаженных символов в кодовой комбинации $g \leq d - 1$

Если $g > d$, то некоторые ошибки также обнаруживаются. Однако полной гарантии обнаружения ошибок здесь нет, так как ошибочная комбинация в этом случае может совпасть с какой-либо разрешенной комбинацией. Минимальное кодовое расстояние, при котором обнаруживаются любые одиночные ошибки, $d = 2$, а минимальное кодовое расстояние, при котором восстанавливается она ошибка на кодируемый символ $d = 3$.

Рассмотрим некоторые простейшие систематические коды, применяемые только для обнаружения ошибок. Одним из кодов подобного типа является код с четным числом единиц. Каждая комбинация этого кода содержит, помимо информационных символов, один контрольный символ, выбираемый равным 0 или 1 так, чтобы сумма единиц в комбинации всегда была четной. Примером могут служить пятизначные комбинации кода Бодо, к которым добавляется шестой

контрольный символ: 10101,1 и 01100,0. Правило вычисления контрольного символа можно выразить в следующей формуле:

$$e = \sum_{i=1}^{k} c_i$$

Отсюда вытекает, что для любой комбинации сумма всех символов по модулю два будет равна нулю. Это позволяет в декодирующем устройстве сравнительно просто производить обнаружение ошибок путем проверки на четность. Нарушение четности имеет место при появлении однократных, трехкратных и в общем, случае ошибок нечетной кратности, что и дает возможность их обнаружить. Появление четных ошибок не изменяет четности суммы, поэтому такие ошибки не обнаруживаются.

К достоинствам кода следует отнести простоту кодирующих и декодирующих устройств, а также малую .избыточность, однако последнее определяет и его основной недостаток — сравнительно низкую корректирующую способность.

6.3.Матрицы

Матрица – это таблица из строк и столбцов

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}_{3\times 2}$$

3×2 –размерность матрицы, здесь сказано, что в матрице А 3 строки и 2 столбца

a_{31} – элемент матрицы, который занимает позицию в 3й строке и 1м столбце

Есть понятия вектор-столбец (размерности)

$$\vec{v} = \begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}_{n\times 1}$$

и вектор-строка (размерности)

$$\vec{u} = \begin{pmatrix} u_1 & \dots & u_n \end{pmatrix}_{1\times n}$$

Очевидно, что матрица А состоит из двух вектор-столбцов или из трех вектор-строк.

Складываются и вычитаются между собой что матицы, что векторы элементарно – покомпонентно (но! Обязательно размерности должны совпадать). Например:

$$A = \begin{pmatrix} 3 & 5 \\ 1 & -1 \\ 3 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & -1 \\ 0 & 4 \\ 9 & 2 \end{pmatrix};$$

$$A + B = \begin{pmatrix} 3+2 & 5-1 \\ 1+0 & -1+4 \\ 3+9 & 1+2 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 1 & 3 \\ 12 & 3 \end{pmatrix}$$

Сложить матрицу с вектором нельзя!

Векторы между собой можно перемножить – это называют скалярным произведением. Нужно покомпонентно премножить и сложить результаты, получим число, которое обозначается так:

$$(\vec{\alpha},\vec{\beta}) = \alpha_1 \cdot \beta_1 + \alpha_2 \cdot \beta_2 + ... + \alpha_n \cdot \beta_n$$

$$\vec{a} = \begin{pmatrix} 3 & 4 & 0 \end{pmatrix},\ \vec{b} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix};$$
$$\vec{ab} = (\vec{a},\vec{b}) = \begin{pmatrix} 3\cdot 1 & 4\cdot 2 & 0\cdot 3 \end{pmatrix} = \begin{pmatrix} 3 & 8 & 0 \end{pmatrix}$$

Матрицы скалярно умножить нельзя! Только векторно и то не всегда

Матрицу A размерностью можно умножить на матрицу B размерностью , только если число столбцов A совпадает с числом строк B (). В результате получится матрица C размерностью (строк как в A, столбцов как в B). Как происходит умножение, проще понять по примеру:

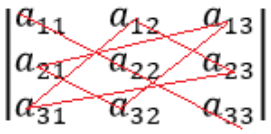
$$A = \begin{pmatrix} 2 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}, B = \begin{pmatrix} 3 & 1 \\ 5 & 4 \\ 7 & 2 \end{pmatrix},$$
$$A \times B = \begin{pmatrix} 2\cdot 3 + 3\cdot 5 + 0\cdot 7 & 2\cdot 1 + 3\cdot 4 + 0\cdot 2 \\ 1\cdot 3 + 0\cdot 5 + 2\cdot 7 & 1\cdot 1 + 0\cdot 4 + 2\cdot 2 \end{pmatrix} = \begin{pmatrix} 21 & 14 \\ 17 & 5 \end{pmatrix}$$
$$B \times A = \begin{pmatrix} 3\cdot 2 + 1\cdot 1 & 3\cdot 3 + 1\cdot 0 & 3\cdot 0 + 1\cdot 2 \\ 5\cdot 2 + 4\cdot 1 & 5\cdot 3 + 4\cdot 0 & 5\cdot 0 + 4\cdot 2 \\ 7\cdot 2 + 2\cdot 1 & 7\cdot 3 + 2\cdot 0 & 7\cdot 0 + 2\cdot 2 \end{pmatrix} = \begin{pmatrix} 7 & 9 & 2 \\ 14 & 15 & 8 \\ 16 & 2 & 4 \end{pmatrix}$$

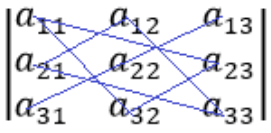
Определитель – число, характеризующее матрицу, рассчитываемое по формулам:

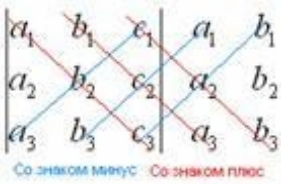
$$\Delta = \begin{vmatrix} a & c \\ b & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\cdot a_{22}\cdot a_{33} + a_{12}\cdot a_{23}\cdot a_{31} + a_{13}\cdot a_{21}\cdot a_{32} -$$
$$-a_{13}\cdot a_{22}\cdot a_{31} - a_{11}\cdot a_{23}\cdot a_{32} - a_{12}\cdot a_{21}\cdot a_{33}$$

Помочь с запоминанием правил могут треугольники или диагонали:







Для любого размера подойдет метод разложения по строке (столбцу)

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} =$$
$$= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31}$$

$$\begin{pmatrix} + & - & + & - \\ - & + & - & + \\ + & - & + & - \\ - & + & - & + \end{pmatrix}$$

Не забывайте чередовать знаки!

Выбирайте строку или столбец, в котором больше нулей или числа меньше по модулю и раскладываем по ней, суммируя такие компоненты: определить знак, взять соответствующий элемент, умножить на минор этого элемента.

$$\begin{vmatrix} 3 & -3 & -5 & 8 \\ -3 & 2 & 4 & -6 \\ 2 & -5 & -7 & 5 \\ -4 & 3 & 5 & -6 \end{vmatrix} = -8 \cdot \begin{vmatrix} -3 & 2 & 4 \\ 2 & -5 & -7 \\ -4 & 3 & 5 \end{vmatrix} + (-6) \cdot \begin{vmatrix} 3 & -3 & -5 \\ 2 & -5 & -7 \\ -4 & 3 & 5 \end{vmatrix} - 5 \cdot \begin{vmatrix} 3 & -3 & -5 \\ -3 & 2 & 4 \\ -4 & 3 & 5 \end{vmatrix} + (-6) \cdot \begin{vmatrix} 3 & -3 & -5 \\ -3 & 2 & 4 \\ 2 & -5 & -7 \end{vmatrix}$$

Минор элемента aij – определитель матрицы, которая получится из первоначальной путем вычеркивания i-й строки и j-го столбца.

Определитель можно вычислить только для квадратной матрицы

Если определитель матрицы равен 0, то матрица – вырожденная.

Дальнейшие параграфы посвящены алгоритмам некоторых помехоустойчивых кодов.

6.4.Код кратных повторений

Наиболее простым и сравнительно более эффективным является код кратных повторений. Идея заключается в повторе каждого бита n раз при отправке. Такой параметр n называют кратностью кода. Например, используя ККП-3 в канал будет отправлено 000000111000 вместо 0010. Очевидно, что кратность напрямую влияет на объем отправляемых данных, сто является и безусловным минусом (страдает скорость передачи информации), так и подспудно – преимуществом (увеличивая избыточность, мы повышаем и помехоустойчивость).

Значительно лучшими корректирующими способностями обладает инверсный код, который также применяется только для обнаружения ошибок. С принципом построения такого кода удобно ознакомиться на примере двух комбинаций: 11000, 11000 и 01101, 10010. В каждой комбинации символы до запятой являются информационными, а последующие — контрольными. Если количество единиц в информационных символах четное, т. е. сумма этих символов равна нулю, то контрольные символы представляют собой простое повторение информационных. В противном случае, когда число единиц нечетное и сумма равна 1, контрольные символы получаются из информационных посредством инвертирования, т. е. путем замены всех 0 на 1, а 1 на 0. При декодировании происходит сравнение принятых информационных и контрольных символов. Если сумма единиц в принятых информационных символах четная, то соответствующие друг другу информационные и контрольные символы суммируются по модулю два. В противном случае происходит такое же суммирование, но с инвертированными контрольными символами. Другими словами, производится r проверок на четность. Ошибка обнаруживается, если хотя бы одна проверка на четность дает 1.

Инверсный код обладает высокой обнаруживающей способностью, однако она достигается ценой сравнительно большой избыточности, которая, как нетрудно определить, составляет величину = 0,5.

6.5.Код Хемминга

Примером исправляющих себя кодов являются – Коды Хемминга – простейшие линейные коды с минимальным расстоянием 3, то есть способные исправить одну ошибку.

Код Хемминга может быть представлен в таком виде, что синдром $\vec{s} = \vec{r}H^T$, где $\vec{r}$ — принятый вектор, будет равен номеру позиции, в которой произошла ошибка. Это свойство позволяет сделать декодирование очень простым.

Для линейных кодов этот метод можно существенно упростить. При этом для каждого принятого вектора $\vec{r_i}$ вычисляется синдром $\vec{s_i} = \vec{r_i}H^T$. Поскольку $\vec{r_i} = \vec{v_i} + \vec{e_i}$, где $\vec{v_i}$ — кодовое слово, а $\vec{e_i}$ — вектор ошибки, то $\vec{s_i} = \vec{e_i}H^T$. Затем с помощью таблицы по синдрому определяется вектор ошибки, с помощью которого определяется переданное кодовое слово. При этом таблица получается гораздо меньше, чем при использовании предыдущего метода.

Помехоустойчивое кодирование называют условно канальным, так как именно в канале происходят процессы, из-за которых мы используем помехоустойчивые коды. Этот вид преобразования призван сохранить целостность информации, в первую очередь от физических помех канала.

Рассмотрим на примере алгоритм Хемминга.

Для кодирования двоичной последовательности необходимо внедрить в нее резервные проверочные биты (пока нулевые), построить порождающую матрицу кода и перемножить матрицу на последовательность, прореженную специальными битами. Полученный синдром расставляется вместо нулей в резервные биты и полученное кодовое слово отправляется в канал.

При получении кодового слова из канала необходимо проверить его на наличие ошибок и снять проверочные биты.

В качестве примера закодируем три байта, отвечающих сообщению "Тат":

11010010 - 11100000 - 11110010

- Внедрим проверочные биты

Резервные биты занимают все ячейки с номерами степеней двойки: 1, 2, 4, 8, 16, …

Заполнение	x	x		x				x			
Номер бита	1	2	3	4	5	6	7	8	9	10	11	12

В оставшиеся ячейки нужно расставить биты передаваемого сообщения:

x	x	1	x	1	0	1	x	0	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12

В резервные позиции временно вставляем нулевые биты:

0	0	1	0	1	0	1	0	0	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12

Этот вектор готов для кодирования.

- Построим порождающую матрицу кода

Матрица для кодирования содержит столько строк, сколько проверочных бит мы используем, и столько столбцов, сколько бит всего в заготовленном векторе. В нашем случае это 4 на 12. Для заполнения матрицы существует несколько мнемонических правил. Например, в каждый столбец можно вносить двоичное разложение его номера снизу вверх

Или можно пользоваться правилом чередования, НО! обязательно не забыть уничтожить нулевой столбец: чередовать нужно 0 и 1 через один символ в первой строке, через 2 - во второй, через 4 - в третьей и т.д.

Построим матрицу чередованием:

Сначала просто выполним чередование. Нужно 4 строки:

```
010101010101010101...
001100110011001100...
000011110000111100...
000000001111111100...
```

Уничтожим столбец нулей и из оставшихся оставим только 12 (столько нам надо):

```
010101010101010101...
001100110011001100...
000111100001111100...
000000001111111100...
```

Итак, построена матрица кода:

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

- Вычислим синдром

Для вычисления синдрома нужно умножить матрицу на построенный вектор-столбец, в результате будет другой вектор-столбец, необходимый для кодирования. Все результаты умножения следует приводить по модулю 2, т.е. ставить 0, если ответ чётный и 1, если нечетный.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = s$$

- Сформируем кодовое слово

Вычисленный синдром следует разместить в резервных битах.

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

1	2	3	4	5	6	7	8	9	10	11	12

Первый байт закодирован: **011010110010**

Закодируйте самостоятельно второй и третий байт и сверьтесь с ответами:

011010110010 - 001011000000 - 001111110010

На этом кодирование завершено

Декодирование кода Хемминга

Допустим, что помехи канала сломали некоторые биты последовательности. Проверим синдромы и декодируем сообщение.

Пусть на первое кодовое слово пришлась одна ошибка в 10 бите, второе кодовое слово осталось без ошибок, а в третьем кодовом слове было две ошибки в 9 и 10 битах. Таким образом, мы получили:

011010110110 - 001011000000 - 001111111110

Для проверки необходимо посчитать синдром кодового слова (умножив проверочную матрицу на полученный вектор) и проанализировать его. Проверочная матрица в данном алгоритме совпадает с порождающей, т.е. соответствует

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

Умножим эту матрицу на первый полученный вектор-столбец. Полученный вектор вновь надо привести по модулю 2. Получили синдром.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = s$$

Если синдром нулевой, гарантированно, что помех в канале не было. В ином случае есть два варианта развития событий: если нам повезло и ошибка была одна, то синдром это двоичное представление номера бита, в котором произошла ошибка. Читать следует снизу вверх. Если не повезло, и ошибок было больше, синдром лишь сигнализирует об этом, но исправить ситуацию самостоятельно уже не выйдет. Остаётся полагаться на везение и декорировать, надеясь, на одну ошибку. Однако не стоит забывать о возможностях кода и наличии большего числа помех.

Итак, в нашем случае синдром оказался не нулевым. Чтобы понять, какой бит нужно

восстановить, переведем в десятичную систему счисления **1010**:

Значит 10й бит испорчен. Мы видим там **1**, значит на самом деле **0**.

Восстановим кодовое слово: **011010110010**

Снимем резервные биты (1, 2, 4, 8, …): **011010110010**

Байт **11010010** соответствует букве "Т" в принятой кодировке.

Рассмотрим два следующих кодовых слова:

Умножим проверочную матрицу на **001011000000**, получим синдром второго кодового слова.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = s$$

Синдром оказался нулевым, значит, в этом кодовом слове гарантированно нет ошибок. Можно снять проверочные биты: **11100000**. Это соответствует букве "а".

Умножим проверочную матрицу на **001111111110**, получим синдром третьего кодового слова:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = s$$

Переведем полученное **0011** в десятичное число:

$$0011_2 \to 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 2 + 1 = 3_{10}$$

Значит 3й бит испорчен. Мы видим там **1**, значит, на самом деле **0**.

Восстановим кодовое слово: **000111111110**

Снимем резервные биты: **000111111110**

Такого байта (**01111110**) вообще нет в принятой кодировке, значит, ошибок в канале явно было больше, чем мог вынести код.

Напомним, что корректирующая способность N определяется для каждого кода по его минимальному расстоянию l_{min}.

$$N = \left\lfloor \frac{l_{min} - 1}{2} \right\rfloor$$

У рассмотренного алгоритма кода Хемминга минимальное расстояние – 3. Поэтому он может исправить только одну ошибку.

Раздел 7. Задачи криптографии. Криптоанализ

7.1. Криптография

Основная цель криптографии – защитить коммуникацию между двумя или более людьми от других людей. Для этой цели она предоставляет средства для уменьшения секретности связи до секретности некоторых данных, также известных как ключи.

Изобретение переменного ключа знаменует рождение криптографии. Первым доказал явную необходимость сокрытия именно ключа уже известный нам Клод Элвуд Шеннон. С тех пор мы различаем «алгоритм» и «ключ». Это также впервые проясняет роль злоумышленника. Основная задача нарушителя информационной безопасности – именно раскрытие ключ, а не прочтение скрытого послания, как могло показаться. Дело в том, что злоумышленник всегда может надеяться на нерадивость и оплошность тех, кто хочет от него защититься в том, что они не будут менять ключи. Так открыв секрет единожды, противник будет читать переписку и впредь. Отсюда возникает очень полезное криптографическое правило – как можно чаще менять ключи.

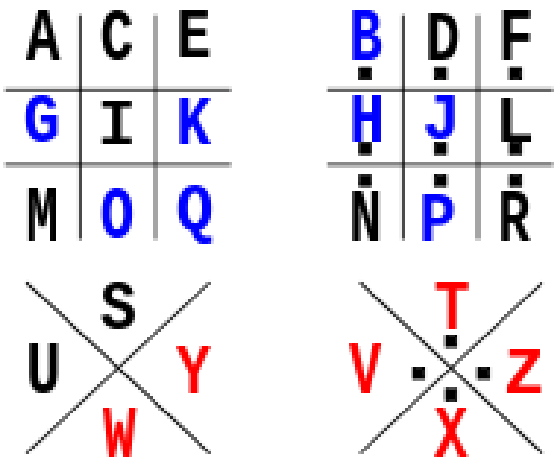
Проще всего это проиллюстрировать с помощью так называемого шифра Цезаря. Его идея приписывается самому Гаю Юлию Цезарю и был развит в 15 веке математиком Леоном Баттистой Альберти. С момента изобретения дисков Альберти эта идея стала неотъемлемой частью криптографических алгоритмов.

Диск Цезаря состоит из большого и малого круглых дисков, которые соединены с возможностью вращения в своих центрах. Алфавит написан на каждом диске в циклическом порядке. Алфавит на внешнем диске называется алфавитом простого текста, а алфавит на внутреннем диске - алфавитом зашифрованного текста. Теперь отправитель и получатель сначала устанавливают определенные настройки для своих целей. Это можно определить, например, указав на внутреннем диске букву рядом с текстовой буквой A. Например, если они согласны с буквой R, диски поворачиваются так, что там, где A находится снаружи, R находится внутри. Вкратце поговорим о «настройке R». Теперь шифрование выполняется таким образом, что обычная текстовая буква заменяется буквой зашифрованного текста, который подключается непосредственно к нему на внутренней панели. Итак, при шифровании вы читаете извне внутрь; соответственно, расшифровка происходит путем чтения изнутри.

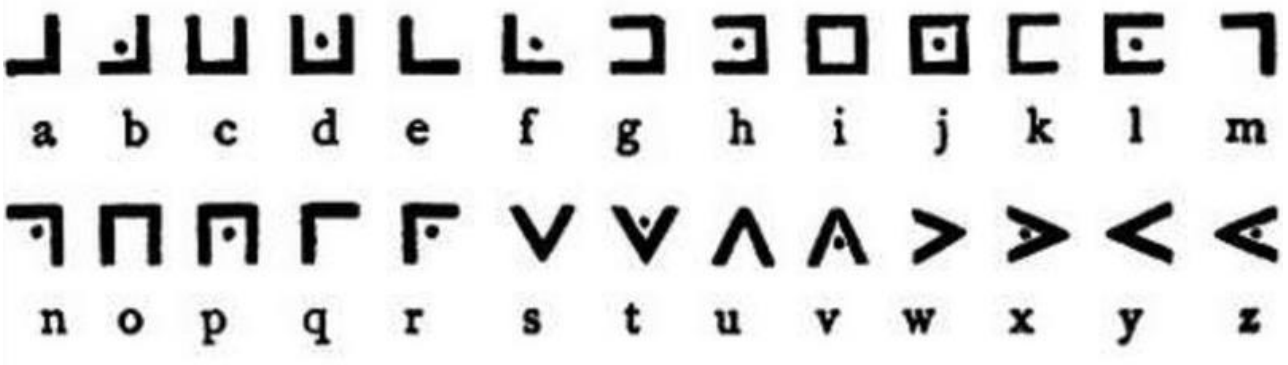
В основном мы можем различать две вещи: во-первых, общий метод шифрования, который также называется алгоритмом шифрования; во-вторых, начальная позиция, в нашем случае первая установка дисков. Это можно использовать для шифрования, то есть преобразования простого текста в зашифрованный текст.

Оригинальный шифр Цезаря гораздо более особенный. Цезарь заменил каждую букву открытого текста буквой, которая стоит в алфавите на три позиции после нее. A становится D, B становится E, а C становится F. CAESAR становится FDHVDU. Очевидно, что это преобразование практически не обеспечивает безопасности, потому что, зная идею алгоритма, нарушителю придется перебрать всего 25 ключей и текст будет прочитан.

Приведем также пример другого красивого исторического шифра – pigpen, или масонский шифр. Для создания ключа принято рисовать такие две схемки и заполнять их буквами в указанном порядке:



Теперь на письме вместо соответствующей буквы ставился символ, который отвечал ее окружению в данной схеме. В виде ключевой таблицы это выглядит так:



Исторически было показано, что хранить в секрете алгоритмы практически невозможно. Основные их принципы неоднократно публиковались или раскрывались, шифровальные машины были украдены или могли быть куплены на законных основаниях. Долгое время изобретатели и разработчики опасались, что это также сделает небезопасным сам процесс коммуникации. Проблема заключалась в следующем: любой, кто знает алгоритм, особенно тот, кто его изобрел, может сломать его. Многие интуитивно понимали, что этого нельзя допускать, но криптолог Огюст Керкхоффс сформулировал это только в 1833 году: осведомленность об алгоритме не должна ставить под угрозу безопасность. Безопасность переписки должна быть основана на том, что ключ хранится в секрете.

Ключ – это некий секрет отправителя и получателя. Так они защищаются от остального мира. Можно также сказать, что ключом является стратегическое преимущество получателя над злоумышленником. Злоумышленник может иметь большой объем зашифрованного текста, он знает процедуру и хочет получить открытый текст. Вопрос в том, сможет ли он это сделать, не зная ключа. Вы также можете изменить ситуацию и измерить безопасность процедуры в соответствии с шансами на успех злоумышленника. Если легко получить открытый текст без ключа, то процесс небезопасен. Это безопасно, когда злоумышленник сталкивается с проблемой, которую он не может решить.

Главное требование к безопасности метода шифрования – это то, что у злоумышленника нет возможности перепробовать все ключи. Теперь очевидно, что шифр Цезаря с 25 ключами точно неактуален сегодня. В современных криптографических алгоритмах ключевое множество должно содержать не менее 2^{128}, а еще лучше 2^{256} элементов. Последнее число больше, чем количество

атомов во Вселенной. В этом отношении очевидно, что никто, даже все компьютеры в Интернете, никогда не сможет перепробовать все возможные ключи для дешифрования.

Важен еще один аспект: с помощью общего ключа вы можете создать пространство для секретного общения не только для двоих, но и для любого количества людей. Если у всех в группе один и тот же ключ и шифрование невозможно взломать, то этот ключ действует как стена, защищающая эту группу от остального мира.

К защите конфиденциальности информации существует всего два подхода: криптография и стеганография. Первая наука занимается сокрытием лишь смысла сообщения в то время, как вторая – сокрытием самого факта существования этого сообщения.

Существует еще одна наука – криптоанализ. Она изучает методы взлома шифров. Вместе криптографию и криптоанализ принято называть криптологией.

Изначально криптография изучала методы шифрования информации – некоего преобразования открытого текста на основе секретного алгоритма или ключа в шифрованный текст с возможностью восстановления. Традиционная криптография разделяется на два класса: симметричную и асимметричную. Симметричные криптосистемы подразумевают единый ключ для двух процессов – шифрования и расшифровывания. Технически ключи могут и не совпадать в точности, но все же если по одному из них просто восстанавливается второй, такой алгоритм все-таки называют симметричным. В асимметричных криптографических протоколах должно быть два разных, связанных ключа. Причем зная один ключ должно быть сложно вычислить второй. Такие системы еще принято называть криптографией на открытом ключе, потому что один ключ публикуется в общем доступе, а второй хранится в секрете, обеспечивая безопасность коммуникации.

Строго говоря в наши дни, криптография уже занимается не только шифрованием, у нее много прочих задач и алгоритмов. К криптографическим процедурам относят и хэширование, и электронно-цифровые подписи, и прочие задачи.

Под алгебраической моделью SA шифра принято подразумевать совокупность пяти множеств:
X – множество открытых текстов,
Y – множество шифртекстов,
K – множество ключей (набор вида $\{k_1, k_2, \dots\}$ для симметричных алгоритмов, и вида $\{(e_1; d_1), (e_2; d_2), \dots\}$ – для асимметричных),
E – множество алгоритмов шифрования,
D – множество алгоритмов расшифрования,
для которых выполнены свойства:
1) инъективности зашифрования: $\forall y \in Y\ \exists x \in X, k \in K: E_k(x) = y$
2) сюръективности расшифрования: $\forall x \in X, k \in K \to D_k\big(E_k(x)\big) = x.$

7.2. Критерии оценивания шифров

Важным параметром любого криптографического алгоритма, помимо стойкости, является его быстродействие.

Для получения объективной оценки быстродействия необходимо использовать независимые от реализации характеристики алгоритмов шифрования, поскольку оценка быстродействия алгоритма в программной реализации сильно зависит от оптимизации программы и ориентации на работу с определенной архитектурой процессора, операционной системой, длиной блока и т.д. Для сравнительной оценки быстродействия разработанных алгоритмов предлагается использовать

методику на основе подсчета количества некоторых элементарных операций, составляющих цикл криптографического преобразования (шифрование 1 блока открытого текста). В результате анализа итеративных блочных шифров в качестве элементарных операций для оценки быстродействия предлагается использовать следующие операции:

- гаммирование (XOR) – g;
- сложение – s;
- умножение – m;
- сдвиг – r;
- табличная подстановка (S-блоки алгоритмов) – t.

Состав используемых для оценки элементарных операций определяется наибольшей распространенностью в существующих блочных шифрах. Для битовой перестановки, встречающейся в некоторых шифрах, в качестве приближенной оценки предлагается использовать операцию сдвига по числу битов перестановки. Операции сложения и умножения выполняются по некоторому модулю, совпадающему с длиной разрядной сетки, необходимой для представления данных, которыми оперирует алгоритм: если алгоритм ориентирован на работу с 16-разрядными данными, то сложение и умножение производится по модулю 216, для 32-разрядных – по модулю 232 и т.д.

В криптографии понятие криптографическая стойкость (криптостойкость) традиционно определяется как устойчивость к попыткам дешифрования сообщений, т.е. количественная мера того, насколько легко криптоаналитик может вскрыть шифр, измеряемая минимальным количеством операций шифрования/расшифрования, необходимых для определения ключа (или открытого текста), при наилучшей из атак. Основное правило криптографии применительно к вопросу криптостойкости состоит в следующем: вскрытие шифра с целью прочесть закрытую информацию должно обойтись гораздо дороже (стоимость выражается в некоторых ресурсах – число микросхем для реализации атаки, объем памяти и т.д.), чем реальная стоимость информации.

Понятие криптостойкости тесно связано со сложностью криптоаналитической атаки на алгоритм шифрования и характеризуется с помощью трех величин:

- сложность по данным - количество перехваченных данных, необходимых для успешной криптоаналитической атаки на алгоритм шифрования.
- вычислительная сложность – количество операций, необходимое для успешной атаки на алгоритм шифрования.
- сложность по памяти - объем памяти, необходимый для успешной атаки.

Для оценки стойкости можно взять максимальную из этих величин или использовать оценку на основе обобщенного критерия.

Все шифры по криптографической стойкости делятся на совершенные (не имеющие более эффективных алгоритмов вскрытия, кроме метода полного перебора) и несовершенные (не обладающие абсолютной стойкостью), стойкость которых основывается на вычислительной сложности решения некоторой математической задачи. Для оценки вычислительной стойкости используется количество операций (w) некоторого типа, необходимых для дешифрования сообщения или определения ключа. Наиболее удобной операцией для получения оценок стойкости является цикл проверки одного ключа. Трудоемкость дешифрования зависит от характера и количества информации, имеющийся в распоряжении аналитика.

Средний объем работы W(N), необходимый для определения ключа по криптограмме, состоящей из N букв, измеренный в элементарных операциях называется рабочей характеристикой шифра. Это значение берется как среднее по всем ключам и всем сообщениям с соответствующими им вероятностями .

Нижние границы трудоемкости достигаются при некоторых конечных значениях параметра N, потому что при его неограниченном увеличении трудоемкость анализа будет стремиться к

некоторому пределу, называемому достигнутой оценкой рабочей характеристики. Как отмечено в [4], для несовершенных шифров не существует способа получить точное значение трудоемкости анализа, все оценки базируются на проверках устойчивости шифров к известным на текущий момент видам криптоанализа, и нет гарантии, что в ближайшем или более отдаленном будущем не будут разработаны новые методы анализа, существенно ее снижающие. Поэтому стойкость несовершенных шифров обосновывается эмпирически как устойчивость к известным на сегодняшний день видам криптоанализа.

7.3. Криптоанализ

Это наука о методах дешифровки зашифрованной информации без предназначенного для этого ключа, а также сам процесс такой дешифровки. В ходе криптоанализа необходимо выяснить ключ (согласно принципу Кирхгоффа, остальное противнику известно).

Основные методы криптоанализа: метод грубой силы, частотный анализ, атака на основе частично подобранных открытых текстов.

Метод грубой силы, он же метод полного перебора, судя по названию состоит в переборе всех возможных ключей до нахождения осмысленной последовательности.

Атака на основе частично подобранных открытых текстов заключается в поиске закономерностей на основе предположений о структуре сообщения (приветствие в начале, подпись – в конце)

1. Анализ на основе только шифртекста - аналитик знает механизм шифрования и ему доступен только шифртекст размером n, что соответствует модели внешнего нарушителя, который имеет физический доступ к линии связи;

2. Анализ на основе заданного открытого текста - криптоаналитику известен шифртекст и та или иная доля исходной информации, а в частных случаях и соответствие между шифртекстом и исходным текстом, т.е. аналитик располагает зашифрованным сообщением размером n и соответствующим ему открытым текстом. Такая атака возможна пришифровании стандартных документов, подготавливаемых по стандартным формам, когда определенные блоки данных повторяются и известны;

3. Анализ на основе произвольно выбранного открытого текста - криптоаналитик может ввести специально подобранный им текст в шифрующее устройство и получить криптограмму, образованную под управлением секретного ключа, т.е. в распоряжении аналитика есть возможность получить результат зашифрования для произвольно выбранного им массива открытых данных размером n, что соответствует модели внутреннего нарушителя, когда в атаку на шифр вовлекаются лица, которые не знают секретного ключа, но в соответствии со своими служебными полномочиями имеют доступ к устройству шифрования;

4. Анализ на основе произвольно выбранного шифртекста - противник имеет возможность подставлять для дешифрования фиктивные шифртексты, которые выбираются специальным образом, чтобы по полученным на выходе дешифратора текстам он мог с минимальной трудоемкостью вычислить ключ шифрования, т.е. в распоряжении аналитика есть возможность получить результат расшифрования для произвольно выбранного им зашифрованного сообщения размером n, что соответствует доступу к устройству дешифрования;

5. Атака на основе адаптивных текстов – атакующий многократно подставляет тексты для шифрования (или дешифрования), причем каждую новую порцию данных выбирают в зависимости от полученного результата преобразования предыдущей порции.

В настоящее время в криптоанализе выделяют следующие основные методы:

1. Полный перебор возможных ключей;

При криптоанализе с известным исходным текстом, который является наиболее актуальным, единственным опубликованным применяемым методом остается полный перебор ключей. Основным недостатком данного метода является экспоненциальная зависимость времени криптоанализа от длины ключа, поэтому, из-за большой трудоемкости этот метод выбран в качестве верхней границы при оценке криптостойкости шифра - если задачу взлома шифра можно решить только методом полного перебора, реализация которого будет неприемлема по финансовым или временным соображениям, то данный шифр считается стойким. Также этот метод используется для оценки эффективности других алгоритмов криптоанализа - все остальные методы должны обеспечивать значительно меньшую трудоемкость.

Существуют оптимизации этого метода, связанные с использованием словарных атак. Идея базируется на том, что отправитель сообщения предпочтет использовать легко запоминаемую ключевую последовательность в отличие от полностью случайной. Поэтому в первую очередь необходимо опробовать наиболее вероятные ключи (и их комбинации) из словаря, что может существенно сократить трудоемкость вскрытия шифра. Практические исследования показали высокую эффективность такого подхода.

2. Частотный анализ – подробнее описан в следующем пункте

3. Дифференциальный метод;

это попытка вскрытия секретного ключа блочных шифров, которые основаны на повторном применении криптографически слабой цифровой операции шифрования (раундовой функции) r раз. При анализе предполагается, что на каждом цикле используется свой подключ шифрования. ДКА может использовать как выбранные, так и известные открытые тексты. Успех таких попыток вскрытия r-циклического шифра зависит от существования дифференциалов (r-1)-го цикла, которые имеют большую вероятность. Идея метода состоит в поиске дифференциалов для последнего цикла, на основании которых можно определить цикловой подключ, после чего процедура повторяется. Отличительной чертой дифференциального анализа является то, что он практически не использует алгебраические свойства шифра (линейность, аффинность, транзитивность, замкнутость и т.п.), а основан лишь на неравномерности распределения вероятности дифференциалов.

4. Линейный метод.

Метод предполагает, что криптоаналитик знает открытые и соответствующие зашифрованные тексты. Обычно при шифровании используется сложение по модулю 2 текста с ключом и операции рассеивания и перемешивания. Метод основан на поиске наилучшей линейной аппроксимации (после всех циклов шифрования) для выражения, описывающего выход шифра, после чего для нахождения каждого бита собственно ключа необходимо решить систему линейных уравнений для известных линейных комбинаций этих бит, но эта процедура не представляет сложности, так как сложность решения системы линейных уравнений описывается полиномом не более третьей степени от длины ключа.

Автоматизация методов криптоанализа для перебора ключей при помощи ЭВМ требует создания адекватной математической модели открытого текста для оценки получаемых результатов. Во многом успех криптоанализа основывается на избыточности открытых текстов на реальных языках [1, 2, 4]. Поэтому перед шифрованием данные целесообразно подвергать преобразованию, уменьшающему избыточность, в качестве которого может выступать сжатие информации. Преимущества использования сжатия перед шифрованием, помимо уменьшения избыточности, позволяет ускорить процесс шифрования, занимающий много времени, за счет сокращения объема открытых данных.

7.4. Частотный анализ

Частотный анализ – основной инструмент для взлома большинства классических шифров перестановки или замены. Данный метод основывается на предположении о существовании статистического частотного распределения символов алфавита. Необходимо посчитать частоту символов в шифртексте и сопоставить их с таблицей вероятностного распределения букв для предполагаемого языка оригинала.

Частотный анализ учитывает частотные характеристики текстов на реальных языках, обладающих большой избыточностью и наличием устойчивых частот сочетаний символов (k-грамм и словоформ языка). Принцип метода состоит в подсчете частот символов в криптограмме и сравнение с вычисленными частотами для модели открытого текста. Достоинством метода является возможность использования для шифров с различающимися алфавитами для открытого текста и криптограммы. Для построения математической модели открытого текста используется построение частотных характеристик открытых текстов, вычисленных с необходимой точностью при исследовании текстов достаточной длины.

Открытый текст рассматривается как реализация стационарного эргодического случайного процесса с дискретным временем и конечным числом состояний и модель представляет собой однородную цепь Маркова [3]. Более высокий порядок приближения соответствует более связному тексту, получаемому на выходе модели. Критерии распознавания строятся либо на основе стандартных методов различения статистических гипотез, либо на основе запретных k-грамм (некоторые редко встречающиеся k-граммы объявляются запретными и их присутствие в получаемом тексте означает получение "бессмысленной" последовательности знаков). При использовании специальных алфавитов (для нетекстовой информации) требуется построение новых формализованных критериев на открытый текст, учитывающих специфику формирования входной последовательности, что само по себе является сложной задачей, связанной с дополнительными исследованиями. Построение критериев для языка, характеризующегося отсутствием статистических зависимостей (равномерным распределением вероятностей появления символов), является практически неразрешимой задачей.

7.5.Классификация шифров

Все шифры принято делить на два класса:

Шифры подстановки. Идея заключается в замене символов открытого текста некоторыми другими символами по указанию ключевой таблицы. Примером может служить рассмотренный ранее шифр сдвига (обобщение шифра Цезаря)

Шифры перестановки. Эти шифры не меняют символы открытого текста, но меняют их позиции, перемешивая по указанию ключа. Примером может служить сцитала, о которой было рассказано ранее

Безусловно для повышения стойкости криптографической системы нужно использовать оба подхода (синтез).

Шифр Альберти подразумевает наличие двух дисков: большого (его называют основным или алфавитным) и малого (его называют шифрующим). На алфавитном диске располагают алфавит (неожиданно, правда?!) сообщения, т.е. все символы, которые, по мнению сторон, должны быть зашифрованы. Эта последовательность должна быть идейно упорядочена. На малом - располагают шифрпоследовательность (те символы, на которые шифратор будет заменять символы сообщения, обычно это те же символы, что и на большом диске, но в перепутанном порядке, хотя и не

обязательно те же символы), обычно ее делают запутанной, чтобы увеличить сложность подбора. Центры дисков следует совместить (советую проткнуть иглой для удобства или придется проверять совпадение центров после каждой зашифрованной буквы - смещение сломает шифр)

Далее необходимо выбрать режим и стартовую позицию. Режимом называют процедуру, которую выполняют после зашифровки каждой буквы. Стандартно принято так: после зашифровки каждой буквы поворачивать диск на 1 позицию по часовой стрелке. (Очевидно, что действие "поворот против часовой стрелки на 3 позиции" не испортит шифрование - это просто другой режим). Стартовая позиция - символ f шифрующего диска, который располагается напротив первого символа g большого диска (т.к. последовательность на основном диске упорядочена по условию, там есть какой-то явно выделенный первый g): f установит шифрсистему для первого символа сообщения, после зашифровки которого малый диск будет повернут по установленному режиму и для второго символа вашего открытого текста шифрсистема уже сменится.

Если было решено не шифровать некие символы открытого текста (т.е. они не помещены на алфавитный диск), то они просто переносятся в шифртекст без изменений, диски при этом не поворачиваются.

Пример.

Допустим, нужно зашифровать свой номер телефона, применив диски Альберти. В этом случае, в алфавит входят числа от 0 до 9. Они по порядку располагаются на основном диске (для удобства, развернем диск в строку и обозначим алфавитный – буквой А). На шифрующий диск (строка под буквой Ш) надо вынести любые символы в том же количестве. Для того, чтобы не привязываться к идее букв, выберем картинки (ведь картинки тоже сойдут за символы).

Итак, что дано. Вот такие диски

А	0	1	2	3	4	5	6	7	8	9
Ш	♈	♉	♊	♋	♌	♍	♎	♏	☯	♑

Режим выберем стандартный (после каждого зашифрованного символа сдвигать Ш на 1 по часовой стрелке, т.е. в терминах строки на 1 позицию вправо). Стартовая позиция – инь-янь, т.е.:

А	0	1	2	3	4	5	6	7	8	9
Ш	☯	♑	♈	♉	♊	♋	♌	♍	♎	♏

Телефон, допустим, начинается с цифр: 89162…

На первой итерации шифруется первый символ сообщения (8), напротив него в А стоит знак ♎ в Ш, значит шифртекст начинается с ♎. Один символ зашифрован, значит сдвигаем Ш:

А	0	1	2	3	4	5	6	7	8	9
Ш	♏	☯	♑	♈	♉	♊	♋	♌	♍	♎

Теперь вторая интеграция и второй символ открытого текста (9). После сдвига напротив А из Ш опять стоит тот же симовл, значит шифртекст теперь – это ♎♎. Вновь сдвигаем Ш:

А	0	1	2	3	4	5	6	7	8	9
Ш	♎	♏	☯	♑	♈	♉	♊	♋	♌	♍

Третья интеграция отвечает за третий символ открытого текста (1). Напротив, 1 из А стоит знак ♏ в Ш, значит шифртекст: ♎♎♏. Сдвиг Ш:

А	0	1	2	3	4	5	6	7	8	9
Ш	♍	♎	♏	☯	♑	♈	♉	♊	♋	♌

Напротив очередного символа (6) в Ш стоит ♉. шифртекст теперь ♎♎♏♉. Сдвиг Ш приводит нас к:

А	0	1	2	3	4	5	6	7	8	9
Ш	♌	♍	♎	♏	☯	♑	♈	♉	♊	♋

Новый символ (2) вновь выпадает на символ ♎. Получится: ♎♎♏♉♎… и так далее.

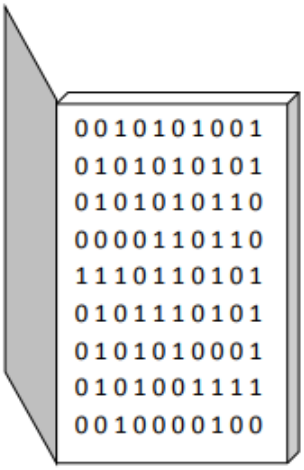
7.6.Абсолютная стойкость шифров

Существуют и абсолютно стойкие шифры – те, что невозможно взломать ныне известными науке способами за разумное время. Математическим определением данного понятия является следующая емкая строка (тут необходимы знания условной вероятности): $\forall x \in X, y \in Y \rightarrow p(x \mid y) = p(x)$. Клод Шеннон доказал существование таких шифров на примере шифра Вернама «Одноразовый блокнот» (возможен для практической отработки). Итак, существуют два критерия абсолютной стойкости: все ключи должны быть равновероятными и случайными, а противнику могут быть известны все детали шифра, кроме ключа.

Шифр «Одноразовый блокнот»: необходимы два совершенно одинаковых блокнота, в которых один к одному записан одинаковый беспорядочный набор нулей и единиц. Действия при шифровании и расшифровании идентичны, ниже представлен алгоритм шифрования одной буквы с помощью блокнота.

Алгоритм шифрования буквы:

№	Действие	Пример
1.	Перевести букву в бинарный код	К → _._ → 101 = ***a***
2.	Из блокнота выписать первые неиспользованные N знаков (N – количество знаков в бинарной записи шифруемой буквы)	N = 3: 001 = ***b***
3.	Произвести «сложение по модулю два»: если знаки разные, то результат равен 1, если одинаковые – 0.	***a*** + ***b*** = ***c*** 1+0=1 0+0=0 1+1=0
4.	Передать код ***c***. N знаков из блокнота, которые использовали для шифра буквы, вычеркнуть.	***c*** = 100



Список используемой литературы

1. Информатика. 10 класс. Базовый и углубленный уровни: учебник: в 2 ч. Ч. 1. ФГОС / К. Ю. Поляков, Е. А. Еремин. – М.: БИНОМ. Лаборатория знаний, 2018. – 352 с.: ил.
2. Информатика. 10 класс. Базовый и углубленный уровни: учебник: в 2 ч. Ч. 2. ФГОС / К. Ю. Поляков, Е. А. Еремин. – М.: БИНОМ. Лаборатория знаний, 2018. – 352 с.: ил.
3. Гашков С.Б. Системы счисления и их применение. – М.: МЦНМО, 2004. — 52 с.
4. Златопольский Д.М. Занимательная информатика. – М.: Бином. Лаборатория знаний, 2011. – 424 с.
5. Лось, А. Б. Криптографические методы защиты информации : учебник для академического бакалавриата / А. Б. Лось, А. Ю. Нестеренко, М. И. Рожков. — 2-е изд., испр. — Москва : Издательство Юрайт, 2018. — 473 с. — (Высшее образование). — ISBN 978-5-534-01530-0. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://urait.ru/bcode/413075 (дата обращения: 27.03.2020).
6. ГОСТ Р 51583 – 2014. Защита информации. Порядок создания автоматизированных систем защищенном исполнении. – Москва: Стандартинформ, 2014. – 18 с.
7. Об информации, информационных технологиях и о защите информации [Текст]: Федеральный закон от 27 июля 2006 г. № 149-ФЗ // Совет Федерации. – 2006.
8. О персональных данных [Текст]: Федеральный закон от 27 июля 2006 г. № 152-ФЗ // Совет Федерации. – 2006.
9. Доктрина информационной безопасности Российской Федерации, утверждена Указом Президента Российской Федерации от 5 декабря 2016 г. №646.
10. Замятина, О. М. Вычислительные системы, сети и телекоммуникации. Моделирование сетей : учебное пособие для магистратуры / О. М. Замятина. — Москва : Издательство Юрайт, 2019. — 159 с. — (Университеты России). — ISBN 978-5-534-00335-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://urait.ru/bcode/433938 (дата обращения: 27.03.2020).