МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

М. І. Романюк, Ю. Г. Савченко

ОСНОВИ ТЕОРІЇ ІНФОРМАЦІЇ ТА КОДУВАННЯ

КОНСПЕКТ ЛЕКЦІЙ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 171 «*Електроніка*»
спеціалізації «Електронні та інформаційні системи і технології
телебачення, кінематографії та звукотехніки»*

Київ
КПІ ім. Ігоря Сікорського
2019

Основи теорії інформації та кодування. Конспект лекцій: [Електронний ресурс]: навч. посіб. для студ. спеціальності 171 «Електроніка», спеціалізації «Електронні та інформаційні системи і технології телебачення, кінематографії та звукотехніки»/ М.І. Романюк; Ю. Г. Савченко; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,86 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. –70 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 7 від 01.04.2019 р.) за поданням Вченої ради факультету електроніки (протокол № 03/2019 від 25.03.2019 р.)

Електронне мережне навчальне видання

ОСНОВИ ТЕОРІЇ ІНФОРМАЦІЇ ТА КОДУВАННЯ

КОНСПЕКТ ЛЕКЦІЙ

Укладачі: *Романюк Маргарита Ігорівна*, канд. техн. наук.
Савченко Юлій Григорійович, професор, докт. техн. наук.

Відповідальний редактор *Лазебний В. С.*, доцент, канд. техн. наук, доцент

Рецензенти: *Буценко Ю. П.,* доцент, канд. фіз-мат. наук

Навчальний посібник містить матеріали дисципліни «Основи теорії інформації та кодування», що викладається студентам спеціальності 171 «Електроніка», спеціалізації «Електронні та інформаційні системи і технології телебачення, кінематографії та звукотехніки». Посібник написано відповідно до навчальної програми вказаної дисципліни та складається з семи тем, кожна з яких розглянута у відповідності з виділеними питаннями плану та на основі наведених рекомендованих інформаційних джерел. Для закріплення теоретичного матеріалу у кінці кожної лекції (теми) представлені контрольні питання.

У посібнику викладені базові поняття сучасної теорії інформації та методи ефективного кодування цифрової інформації в комп'ютерних та телекомунікаційних системах з точки зору захисту від завад та забезпечення оптимальної швидкості інформаційного обміну. Матеріали посібника будуть корисними студентам, магістрантам, викладачам, а також усім зацікавленим.

ЗМІСТ

ВСТУП

При вивченні дисципліни «Теорія інформації та кодування» необхідно зважати, що предметом вивчення є об’єкти, які характеризуються кількісними параметрами та математичними співвідношеннями, що їх зв’язують. Тому семантичний (змістовний) характер цих об’єктів враховувати не потрібно. Термін «Інформація» є одним з тих термінів, які досить часто зустрічаються не тільки в наукових працях спеціального характеру, але й у безлічі повсякденних ситуацій і є інтуїтивно зрозумілим кожній людині. Однак у різних галузях знань у нього вкладають різний зміст. Різноманітність інформаційних процесів та великий інтерес до них у різних областях діяльності людей породили багато тлумачень поняття «інформація», а отже, і визначення кількості інформації.

Термін «інформація» походить від латинського слова “informatio”, що означає «роз’яснення» і, по суті, передбачає наявність деякого діалогу між відправником та отримувачем інформації. Таким чином, одне з ключових питань, з якого існують різні точки зору, полягає в тому, що інформація – це властивість об’єкту чи результат взаємодії. Ми будемо притримуватися точки зору А.М. Колмогорова: інформація існує незалежно від того, сприймається вона чи ні, але проявляється тільки при взаємодії.

Факт об’єктивного існування інформації незалежно від нашої свідомості дає підстави багатьом вченим стверджувати, що інформація поряд з матерією і енергією є первинним поняттям нашого світу, не зважаючи на те, що для неї поки що не сформульовані фундаментальні закони збереження і перетворення в матерію і/або енергію. Разом з тим, потрібно підкреслити, що інформація завжди проявляється у матеріально-енергетичній формі у вигляді сигналів, хоча це не матерія і не енергія, які переходять одна в одну. Інформація може зникати і з’являтися. Таким чином, інформація в строгому сенсі не може бути визначена. Можна лише перерахувати її основні властивості, наприклад такі як:

• інформація приносить відомості про навколишній світ, яких в даній точці (або в даний момент часу) не було до її отримання;

• інформація не матеріальна, але вона виявляється у формі матеріальних носіїв дискретних знаків або первинних сигналів;

• знаки і первинні сигнали несуть інформацію тільки для одержувача, здатного їх розпізнати.

Метою вивчення дисципліни «Основи теорії інформації та кодування» є формування у студентів комплексу знань, умінь, навичок, необхідних для розуміння основних положень теорії інформації, а також понять, структур, принципів дії систем кодування інформації, що ґрунтується на засадах аналогової та цифрової техніки, техніки обробки та передачі інформації.

Набуті при вивчені дисципліни «Основи теорії інформації та кодування» можуть бути використані під час вивчення інших дисциплін, тісно пов’язаних з теорією інформації та кодуванням.

У навчальному посібнику викладені основні питання, що стосуються проблем кодування (стиснення) повідомлень дискретних ймовірнісних джерел, а також основні поняття, які зв'язані з передачею інформації через канал зв'язку та кодами, що виправляють помилки.

ТЕМА 1. ОСНОВНІ ПОНЯТТЯ ТА ВИЗНАЧЕННЯ

Термін «Інформація» будемо розуміти у вузькому сенсі, який використовується для опису так званих інформаційних систем.

Інформаційні системи – це клас технічних систем, які призначені для зберігання, передавання та перетворення інформації. Ми обмежимося розглядом дискретних систем передавання інформації. Це охоплює процеси передавання інформації у комп'ютерних системах, цифрових лініях зв'язку та мережах. Відповідно інформація – це сукупність відомостей про навколишній світ, які є об'єктом зберігання, передачі та перетворення.

Отже, основна проблема, яку необхідно розв'язати при побудові систем передавання та зберігання інформації, була вперше сформульована Клодом Шеноном у 1948 році: «Основна властивість системи зв'язку полягає в тому, що вона повинна точно або приблизно відтворити у певній точці простору та часу деяке повідомлення, яке було сформоване в іншій точці. Природно, це повідомлення має певний зміст, однак це зовсім не важливо для сформульованої інженерної задачі. Найголовніше полягає в тому, що надіслане повідомлення вибирається з деякого сімейства можливих повідомлень».

Така точна й зрозуміла постановка проблеми зв'язку справила величезний вплив на розвиток засобів зв'язку. Виникла нова наукова галузь, яка стала називатися теорією інформації – розділ кібернетики, який займається розробкою математичних методів для передавання, збереження та перетворення інформації. Головна ідея, обґрунтована К.Шеноном, полягає в тому, що надійна система зв'язку повинна бути цифровою, тобто задачу зв'язку можна розглядати як передачу двійкових цифр.

Відзначимо, що будь-який фізичний канал передачі сигналів не може бути абсолютно надійним. Це пов'язано з дією шуму, який псує канал та вносить помилки в цифрову інформацію, що передається.

К. Шенон показав, що при виконанні деяких достатньо загальних умов є принципова можливість використовувати ненадійний канал для передавання інформації з як завгодно великим ступенем надійності. Тому немає необхідності робити спроби очищення каналу від шумів, наприклад, підвищуючи потужність сигналів (це коштовно, а іноді буває навіть неможливо). Замість цього потрібно розробляти ефективні схеми кодування та декодування цифрових сигналів. Крім того, К. Шенон довів, що задачу надійного зв'язку можна розкласти на дві підзадачі, не зменшуючи її ефективність. Ці дві підзадачі називаються кодуванням джерела і кодуванням каналу.

Задачею кодування джерела є побудова кодера джерела, який продукує компактний опис повідомлення джерела, який необхідно передати адресату.

Джерело інформації створює повідомлення або послідовність повідомлень, які необхідно передати. Задача кодування каналу полягає в тому, що за відомими характеристиками каналу будується кодер каналу, який посилає в канал вхідні символи, що будуть відтворені (декодовані)

Каналом може слугувати пара мідних дротів, коаксіальний кабель, оптоволоконний кабель, супутникова антена, оптичний диск, пам'ять комп'ютера та ін. Фізичне поняття каналу зв'язку у теорії інформації замінюється деяким абстрактним поняттям системи обмежень, які діють на сигнали у процесі передавання. Переважно такі обмеження пов'язані з дією завад, або, по-іншому, наявності шуму в каналі, що спотворює сигнал передавача.

Будь-який матеріальний об'єкт разом із спостерігачем утворює систему, яка називається джерелом повідомлень. Якщо стан матеріального об'єкту, а отже, і його фізичні показники можуть набувати значення з певного дискретного набору значень, то джерело повідомлень з таким об'єктом є дискретним.

Отже, дискретне джерело інформації – це таке джерело, яке може виробити (згенерувати) за скінчений відрізок часу тільки скінчену множину

повідомлень. Кожному такому повідомленню можна співставити відповідне число (код), та передавати його замість повідомлень.

Якщо стан матеріального об'єкту набуває значення з нескінченної множини можливих значень, то таке джерело повідомлень є неперервним. Принципово воно може бути зведене до дискретного, якщо прийняти допустимий рівень похибки та за її допомогою з нескінченної множини можливих значень повідомлень вибрати певний дискретний набір. Саме тут у наявності похибки та її допустимому рівні і криється принципова різниця між дискретним і неперервним джерелами повідомлень. Для переведення неперервної інформації в дискретну і навпаки використовуються спеціальні пристрої, що здійснюють дискретизацію та квантування сигналів, модуляцію/демодуляцію (модеми), тощо. Таке перетворення виконується в більшості сучасних телекомунікаційних систем та систем управління.

Якщо під час деякого часового проміжку дискретним джерелом вибрано деяке повідомлення x_i, яке ніяк не зумовлене повідомленням, отриманим у попередній проміжок часу, то таке джерело є дискретним *джерелом без пам'яті.*

Якщо в деякому часовому проміжку дискретним джерелом вибрано повідомлення, зв'язане з попереднім повідомленням і статистично зумовлене ним, то таке джерело називається дискретним *джерелом із пам'яттю*.

Якщо у кожний проміжок часу дискретне джерело повідомлень вибирає одне з k можливих повідомлень $1,2...x_k$, то кажуть, що $1,2...x_k$ є дискретною множиною повідомлень, або алфавітом джерела повідомлень X, а k - об'єм алфавіту джерела або потужність джерела. Як правило, для повнішого опису джерела повідомлень на множині X визначають його ймовірнісну міру, тобто з кожним дискретним повідомленням x_i співставляють ймовірність p_i вибору джерелом. Отже, множині $1,2...x_k$ співставляється ймовірнісна міра у вигляді множини $1,2,\ldots p_k$, на яку накладено обмеження у вигляді $\sum_{1}^{k} p_i = 1.$

Якщо всі ймовірності, які визначають виникнення символів на виході джерела, не залежать від часу, джерело називають стаціонарним. Ми будемо розглядати тільки стаціонарні джерела та для скорочення замість “стаціонарне джерело” будемо всюди використовувати просто “джерело”.

Дві множини X та $P(x_i)$ дають достатньо повний опис дискретного джерела повідомлень у вигляді його ймовірнісної моделі, а тому разом вони утворюють ансамбль повідомлень дискретного джерела. Наведене вище обмеження є природною умовою включення до складу множини X повної групи подій, якими виступають дискретні повідомлення, і це дає змогу врахувати при розгляді всі події-повідомлення.

Більшість реальних джерел інформації є джерелами з пам’яттю. Розподіл ймовірностей виникнення чергового символу на виході дискретного джерела з пам’яттю залежить від того, які символи були попередніми. Таке джерело інформації називають марковським, оскільки процес появи символів на його виході адекватний ланкам Маркова.

Будемо говорити, що глибина пам’яті марковського дискретного джерела інформації дорівнює h, якщо ймовірність появи чергового символу залежить тільки від h попередніх символів на виході цього джерела.

Припустимо, що стан деякого об’єкта або системи наперед відомий. Тоді повідомлення про цей стан не несе ніякої інформації для її одержувача. Якщо ж стан об’єкта змінився і джерелом передане якесь інше повідомлення про стан об’єкта, то це повідомлення несе нові відомості, які додадуть знання про об’єкт. Тоді можна говорити, що таке повідомлення містить деяку кількість інформації для її одержувача.

В основі теорії інформації лежить запропонований Клодом Шеноном спосіб вимірювання кількості інформації як випадкової величини відносно іншої випадкової величини. Цей спосіб дозволяє виразити кількість інформації числом і надає можливість об’єктивно оцінити інформацію, що міститься у повідомленні.

Отже, вважаємо, що повідомленнями є випадкові події (принаймні з точки зору отримувача повідомлення) і нехай дискретне джерело інформації видає послідовність повідомлень, кожне з яких вибирається з алфавіту 1,2... x_k з певною ймовірністю $p(x_i)$, а k - об'єм алфавіту джерела.

Для такого дискретного джерела кількість інформації визначається за таким виразом

$$H = -\sum_{1}^{k} p_i \log_2 p_i \ \textit{біт/символ}, \qquad (1.1)$$

причому основа логарифма може бути довільною. Найчастіше основу логарифма вибирають рівною 2. У цьому випадку за одиницю кількості інформації беруть біт.

Отже, біт – це кількість інформації у повідомленні дискретного джерела, алфавіт якого складається з двох альтернативних подій, які є апріорно рівноймовірними. Легко довести, що коли всі повідомлення мають однакову ймовірність, тобто є рівноймовірними, формула (1) спрощується до

$$H = \log_2 k, \qquad (1.2)$$

і це є *максимальне* значення кількості інформації, яке можна отримати від джерела.

Зазначимо, що існують інші одиниці вимірювання кількості інформації. Так, якщо б логарифм обчислювався за натуральною основою, то одиницею кількості інформації був би нат (nat, natural digit), а якщо основа логарифма десяткова, то кількість інформації вимірюється у хартлі (у честь Р. Хартлі (Hartley), який використовував формулу ще до К. Шенона).

Вимірювання кількості інформації у бітах найбільш зручно, оскільки вони відразу показують, скільки двійкових символів необхідно витратити на передачу повідомлення. Але у теоретико-інформаційних дослідженнях часто надають перевагу натам, оскільки $ln(x)$ зручніше диференціювати ніж $log(x)$.

Контрольні питання

1. Дайте визначення поніттю «інформація. Які вам відомі форми прояву інформації?
2. У чому полягають задачі кодування джерела та кодування каналу?
3. Назвіть основні відмінності для джерела без пам’яті та джерела з пам’яттю. Що таке дискретне джерело інформації?
4. Від чого залежить кількість інформації, яка припадає на одне повідомлення джерела?
5. Який принцип вимірювання інформації використовують у теорії інформації? Поясніть сутність цього принципу.
6. Які одиниці виміру інформації вам відомі? Що таке біт?
7. Як визначити кількість інформації дискретного джерела повідомлення?

ТЕМА 2 ІНФОРМАЦІЙНА НАДЛИШКОВІСТЬ ЯК УНІВЕРСАЛЬНИЙ ЗАСІБ КОНТРОЛЮ ПЕРЕДАВАННЯ ТА ЗБЕРЕГАННЯ ІНФОРМАЦІЇ

Традиційно поняття інформаційної надлишковості (ІН) найчастіше пов'язують із використанням завадозахищених кодів для передачі і зберігання інформації. За К. Шеноном рівень ІН визначається відносним перевищенням максимально можливої ентропії H_{max} над реальною ентропією $H_{реал}$ конкретного джерела інформації при використанні певного способу кодування

$$J = 1 - \frac{H_{реал}}{H_{max}}, \qquad (2.1)$$

де $H_{max} = log_2 N$ та $H_{реал} = \sum_{i=1}^{N} p_i \, log \, p_i$, а N – кількість можливих повідомлень, p_i - ймовірність використання (появи) i-го повідомлення.

Таке перевищення виникає при будь-якому відхиленні розподілу ймовірностей появи окремих повідомлень від рівномірного (при кодуванні повідомлень комбінаціями символів однакової довжини). У частковому випадку, коли окремі комбінації джерелом не використовуються (ймовірність їх появи дорівнює 0), множина всіх можливих повідомлень X_0 природно розпадається на дві підмножини: підмножину дозволених слів X_g (слів, які використовуються джерелом), та підмножину заборонених слів $X_з$ (слів, які джерелом не використовуються). Цей випадок із практичної точки зору є найбільш цікавим, про що піде мова нижче.

Якщо підходити до ІН менш формально, то її присутність проявляється у специфічності, точніше, індивідуальності джерела. При використанні завадозахищених кодів ця індивідуальність досягається додаванням перевірочних сигналів, утворених за певними правилами. Виконання саме цих правил і робить джерело індивідуальним, тобто таким, яке можна впізнати і відрізнити від інших, якщо ті використовують інші способи кодування.

Перевірка виконання штучно введених правил дозволяє виявляти та (або) виправляти помилки, які виникли при передачі чи зберіганні інформації.

Але наявність ІН не є обов'язковою для того, щоб джерело було індивідуальним. Адже навіть у випадку, коли всі повідомлення рівноімовірні ($H_{max} = H_{реал}$ та $J = 0$), джерело залишається специфічним. Його індивідуальність саме в тому і полягає, що всі повідомлення рівноімовірні, а відхилення від рівноімовірності може свідчити про наявність помилок при передачі або збереженні інформації. Цю ситуацію можна вважати навіть парадоксальною: жодної надлишковості немає, а помилки, принаймні деякі, можна виявити.

Таким чином, можна вважати, що будь-яке джерело з відомою та стаціонарною статистикою повідомлень (навіть джерело білого шуму) є специфічним. А якщо цю специфіку можна визначити за допомогою деяких формальних правил, то можна й виявити помилки. Із цього витікає, як наслідок, що контроль достовірності будь-якого джерела можна здійснити без введення штучної надлишковості. Таке ствердження не є результатом формального доведення. Це, скоріше, припущення, на користь якого можна навести безліч прикладів. Саме це ми й зробимо у подальшому.

1. Повідомлення генеруються джерелом із різною ймовірністю, розподіл довільний, заданий спектрограмою, отриманою, наприклад, як результат тривалого спостереження за повідомленнями джерела. Така спектрограма зображує відносні частоти появи того чи іншого слова (символу). Відомо, наприклад, що в англійській мові символ (літера) *E* з'являється із відносною частотою (ймовірністю) 0,12, символ *W* - 0,02 і т.д. В українській – розподіл інший. Найбільшу частоту мають літери *О* (0,08) та *А* (0,07), а найменшу - *Ц* (0,0044) та *Є* (0,0037).

Саме розподіл частот характеризує індивідуальність відповідного джерела. Це – як відбитки пальців людини, за якими її можна "впізнати", тобто ідентифікувати.

З точки зору контролю частотний розподіл можна вважати досить надійним критерієм достовірності інформації. Будь-яке суттєве відхилення розподілу від зафіксованого за тривалий час спостереження є ознакою виникнення помилок. Ясно, що поодинокі відхилення (одиночні помилки) навряд чи призведуть до порушення розподілу і тому не можуть бути надійно виявлені. Це, зрозуміло, є принциповою вадою всіх без винятку статистичних методів контролю. Більш того, навіть у випадках, коли помилки мають регулярний характер, їх можна виявити тільки із запізненням, знову ж таки за тривалий час спостереження. Із цього витікає, що безпосередньо статистичний метод контролю не може бути оперативним, і тому він непридатний для контролю у випадках, коли отримана від джерела інформація зразу ж використовується для управління реальними об'єктами, коли помилки, навіть одиночні, можуть виявитися неприпустимими з точки зору можливих наслідків і втрат.

2. У частотному розподілі є "провали" – деякі повідомлення (слова або символи) мають нульову ймовірність при нормальному функціюванні об'єкта контролю (джерела). Вони утворюють підмножину заборонених слів $X_з$. Поява будь-якого слова із $X_з$ є ознакою помилки, яка може бути *виявлена практично миттєво* апаратними чи програмними засобами. Для цього достатньо знати склад $X_з$. Враховуючи, що підмножина дозволених слів X_g та $X_з$ взаємно доповнюють одна одну до множини всіх можливих слів X_0, тобто

$$X_0 = X_g \cup X ,$$

можна обмежитись фіксацією лише однієї підмножини. Звичайно, для такої фіксації та реалізації відповідних процедур контролю необхідні вільні обчислювальні ресурси, насамперед, пам'ять. Але головна проблема не в ресурсах, а в часових обмеженнях, адже процедуру контролю необхідно виконувати кожного разу при надходженні чергової порції повідомлень, а для цього необхідний час. Тому процедури контролю мають бути максимально

короткими за часом і простими. На цій проблемі ми зупинимось трохи далі, а зараз звернемо увагу на таке.

З теоретичної точки зору між ситуаціями 1 і 2 немає принципової різниці. І в першому, і в другому випадку властивості джерела описуються однаково – довільним частотним розподілом, і цей розподіл можна використати як еталон для виявлення помилок. Але з точки зору практичної реалізації та ефективності контролю за критерієм мінімуму часу, за який помилка може бути виявлена, друга ситуація принципово відрізняється від першої. Саме наявність слів із нульовою ймовірністю дає можливість виявляти відповідні помилки практично в момент їх появи. Виникає питання: а що робити в ситуації 1? Чи можна дати собі раду у цьому випадку? Виявляється, що так. Вихід може підказати все таж класична робота К. Шенона. Цей вихід дуже простий і прозорий. Якщо розглядати частотний розподіл, утворений парами слів, які надходять від джерела у сусідні моменти часу, то майже напевне виявляється, що деякі пари мають нульову ймовірність надходження. Це свідчить, що ми автоматично переходимо до ситуації 2 і отримуємо можливість оперативного виявлення помилок. Якщо ж ми перейдемо до розподілу “”трійок”, “четвірок” і т.д. сусідніх за часом повідомлень, то кількість таких штучно об’єднаних слів, що мають нульову ймовірність, буде зростати у геометричній прогресії.

Для ілюстрації можна навести приклад повідомлень, що надаються, наприклад, українською мовою. На рівні окремих літер у частотному розподілі немає провалів – у текстах використовуються всі літери, але з різною ймовірністю. Але вже в розподілі пар такі провали з’являються: м’який знак не може використовуватися після голосних та на початку слова, не використовуються літеросполучення “дє”, “кє” та інші сполучення літери “є” з приголосними та голосними.

Це були, так би мовити, лінгвістичні приклади, які начебто не мають відношення до контролю електронних приладів та комп’ютерних систем. Але ж, якщо уважно проаналізувати, наприклад, повідомлення від конкретного

давача в комп'ютерну систему управління технологічним об'єктом, то можна зауважити певні закономірності, що мають місце у статистичній структурі таких послідовностей. Наприклад, якщо це значення температури рідини, яку нагрівають, то ці значення будуть зростати, і кожне значення температури в момент часу t буде не менше, ніж в момент часу $t-1$. Або, наприклад, тиск у об'ємі, що зменшується, не може теж зменшуватися, бо відповідно до закону Бойля-Маріота добуток тиску на об'єм повинен залишатися постійним, $PV = const$. До речі, саме ця залежність може бути використана у відповідних випадках як надійна контрольна ознака достовірності даних, що надходять від об'єкта в систему управління. У багатьох інших випадках саме "зв'язаність" деяких параметрів фізичними законами може виявитися найбільш корисною для організації ефективного контролю.

Ще один приклад. В типових комп'ютерних системах промислового призначення більша частина інформації надходить до системи шляхом регулярного опитування давачів за певним (фіксованим) часовим регламентом. Як наслідок, повідомлення від окремих давачів, що характеризують параметри одного технологічного процесу, обов'язково будуть корельовані тому, що це параметри одного і того ж процесу, який не може виходити за межі відповідних фізичних (хімічних) законів. Тому, як і в попередньому випадку, виконання цих законів може бути ефективно використано для перевірки достовірності даних. Зауважимо, що саме корельованість даних свідчить, про наявність *ІН* , тому в таких випадках введення штучної *ІН* не є обов'язковим. І головне, контроль на основі *ІН* , яку логічно назвати *природною*, охоплює не тільки помилки засобів передачі даних (як це має місце при застосуванні штучної *ІН*), а й увесь автоматизований комплекс, тобто порушення технологічного процесу та несправності засобів автоматизації. А це найбільш суттєво з точки зору загальної ефективності контролю та запобігання аварійних ситуацій.

Повернемось тепер до більш конкретних задач, пов'язаних з реалізацією контролю на основі природної *ІН*. Можна сподіватись, що наведені

розміркування і приклади досить переконливо свідчать на користь існування майже у всіх цікавих з практичної точки зору випадках природної *ІН*. Таким чином, залишається основне питання: як цю надлишковість використати для організації контролю інформаційного процесу. Для цього необхідно розв'язати, принаймні дві задачі.

По-перше, бажано попередньо оцінити ефективність контролю щодо його повноти, тобто визначити, наприклад, яку частину всіх можливих помилок можна принципово виявити при його застосуванні. По-друге, потрібно побудувати основні процедури, що дозволяють реалізувати контроль у складі існуючих або таких, що проектуються, телекомунікаційних та комп'ютерних систем.

Перша задача може бути розв'язана на основі досить простих міркувань. Розглянемо випадок, коли джерело (давач, первинний перетворювач, клавіатура, з якої оператор вводить виробничу інформацію в систему, тощо) формує повідомлення у вигляді двійкових слів фіксованої розрядності n. Тоді загальна кількість усіх можливих помилок (векторів помилок) може бути записана як $(2^n - 1)$.

Вектор помилки – це двійкова n-розрядна комбінація, в якій 1 відповідають тим розрядам, що створені помилками. Наприклад, якщо правильна комбінація

$$X_i = (10110010),$$

а після спотворення помилкою отримали

$$X_i' = (10010110),$$

то відповідний вектор помилки

$$E = X_i \oplus X_i' = (00100100)$$

де $\oplus$ – порозрядна сума по модулю 2. Для наочності цю операцію можна записати так

$$\begin{array}{r} X_i = 10110010 \\ \oplus \quad X_i' = 10010110 \\ \hline E = 00100100 \end{array}$$

З іншого боку, очевидно, можуть бути виявлені тільки такі помилки, які переводять (перетворюють) дозволене слово в заборонене. Тому кількість помилок, що можуть бути виявлені, у точності дорівнює кількості заборонених слів (позначимо цю кількість через $m = (X_з)$). Тоді частина помилок, які можна виявити

$$\delta = \frac{m(X_з)}{2^n - 1}, \qquad (2.2)$$

Висновок із цього важливого співвідношення очевидний: чим менше слів (комбінацій) із числа, всіх можливих використовується, тим більше помилок можна виявити. Але цей висновок занадто загальний, щоб ним скористатися конструктивно, тобто для того, щоб визначити конкретні помилки, які можна виявити, та побудувати відповідні процедури корекції.

Тут необхідно зазначити, у загальному випадку, очевидно, не всі можливі помилки реально можуть виникнути, і крім того, з тих помилок, які можуть реально виникнути, не всі однаково небезпечні. Тому має сенс диференціювати елементи підмножини $X_з$ у відповідності з двома видами помилок: помилки, які обов'язково повинні бути виявлені, і помилки, які корегувати не обов'язково. Мова йде, фактично, про скорочення кількості помилок, які підлягають виявленню, враховуючи тільки такі помилки, які найбільш ймовірні або найбільш небезпечні з точки зору функціонування об'єкту. Таке скорочення дозволяє спростити відповідні процедури контролю, що дуже важливо при їх реалізації.

В деяких випадках рівень IH такий, що дозволяє не тільки виявляти, але й виправляти помилки. Розглянемо, за яких умов це можливо. Для наочності скористаємося графічним представленням помилок у вигляді переходів між повідомленнями, або, точніше, перетворень дозволених слів у заборонені (рис. 1).

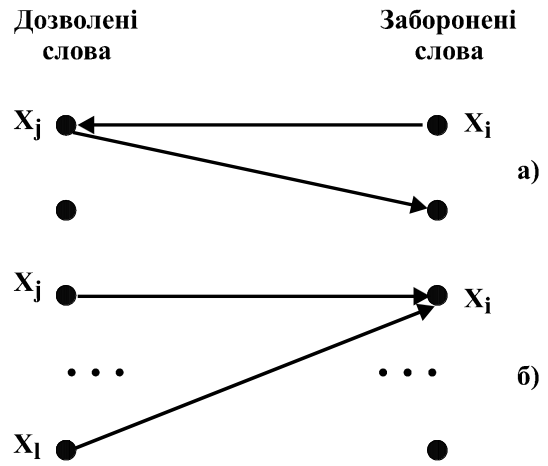

Рисунок 2.1 – Перехід дозволених слів у заборонені

Очевидно, необхідною умовою можливості виправлення конкретної помилки є однозначність зворотного переходу від слова X_i, яке містить помилку, до слова без помилок X_j. Очевидно, що така однозначність має місце лише у випадку, коли в кожне заборонене слово переходить не більше одного дозволеного слова (ситуація а). Коли ж в одне заборонене слово переходить кілька дозволених (ситуація б), визначити "зворотний шлях" для виправлення помилки практично неможливо, не маючи підстав для того, щоб віддати перевагу одному із варіантів зворотного переходу. Виходячи із таких простих розміркувань, можна стверджувати, що для виправлення $m(E)$ помилок у кожному дозволеному слові повинно існувати $m(E)$ відповідних заборонених слів. Для всіх дозволених слів джерела можна записати

$$m(X_g)\cdot m(E) \le m(X_з),$$

звідки

$$m(E) \le \frac{m(X_з)}{m(X_g)} \quad . \qquad (2.3)$$

Отримане співвідношення визначає рівень IH, необхідний для виправлення $m(E)$ векторів помилок при довільному способі кодування, і може слугувати критерієм для попередньої оцінки потенційної корегуючої здатності того чи іншого способу представлення інформації відповідним

джерелом. Як ми побачимо далі, ця формула має універсальний характер і може бути застосована і у випадку штучної IH, наприклад, при використанні завадостійких кодів.

Контрольні питання

1. У чому проявляється наявність інформаційної надлишковості.
2. Що таке “природна” інформаційна надлишковість? Наведіть приклади її присутності.
3. Чим визначається здатність до виявлення або виправлення помилок?
4. Яким чином можна штучно збільшити рівень інформаційної надлишковості?

ТЕМА 3. СТАТИСТИЧНІ МЕТОДИ ЕКОНОМНОГО КОДУВАННЯ

Мета економного кодування даних або кодування джерела – подати дані для передавання через канали зв’язку у максимально компактній та неспотвореній формі. Таке кодування є ефективним для передавання інформації по каналах без завад, причому, збільшення ефективності передавання досягається завдяки вилученню надлишковості у кодах, що передаються. При кодуванні нерівноймовірних повідомлень для зменшення надлишковості коду, необхідно зменшувати середню довжину кодових слів. З іншої сторони, вимога префіксності коду накладає жорсткі обмеження на довжини кодових слів і не дає можливості вибирати кодові слова надто короткими. У зв’язку з цим було запропоновано використовувати так звані префіксні коди, довжина кодових комбінацій яких узгоджувалася б з ймовірністю появи символів у повідомленні. Таке кодування називається статистичним.

Нерівномірний код при статистичному кодуванні вибирається так, щоб більш ймовірні символи передавалися за допомогою більш коротких комбінацій коду, а менш ймовірні – за допомогою більш довгих. У результаті зменшується середня довжина коду. Потенційні можливості ефективного кодування визначає теорема Шеннона, яка для двійкових кодів формулюється так: для будь-якого дискретного джерела інформації існує такий спосіб кодування, що середня довжина кодової комбінації двійкового коду в розрахунку на один символ джерела буде як завгодно близькою до ентропії цього джерела, вираженої в бітах, але не може бути меншою за неї.

Як уже відзначалося, однозначне декодування прийнятих повідомлень забезпечують префіксні коди. У такому коді кожний символ у векторі даних, що кодується, замінюється кодовим словом з префіксної множини двійкових послідовностей або слів. Префіксною множиною двійкових послідовностей S називається скінченна множина двійкових слів, в якій жодна з послідовностей

не є префіксом (початком) ніякої іншої. Наприклад, множина двійкових слів S={0,10,1100,1101,1110,1111} є префіксною множиною двійкових послідовностей.

Отже, якщо необхідно закодувати повідомлення у вигляді деякого вектора то кодування повідомлення кодом без пам'яті здійснюється у такий спосіб:

1) складається повний список символів алфавіту за спаданням їх частот появи у повідомленні;

2) кожному символу призначається кодове слово з префіксної множини S двійкових послідовностей;

3) виходом кодера є об'єднання в одну послідовність усіх отриманих кодових слів.

Вимога префіксності накладає жорсткі умови на довжини кодових слів. Формально ці обмеження записуються у вигляді нерівності, яка називається нерівністю Крафта

$$\sum_{i=1}^{k} 2^{-l_i} \leq 1$$

Якщо нерівність переходить в строгу рівність, то такий код називається компактним і має найменшу серед кодів з даним алфавітом довжину, тобто є оптимальним.

Приклади префіксних множин і відповідних їм векторів Крафта: S={0, 10, 11}→ ν(S1)=(1, 2, 2); 2S={0, 10, 110, 111}→ ν(S2)=(1, 2, 3, 3); 3S={0, 10, 110, 1110, 1111}→ ν(S3)=(1, 2, 3, 4, 4); 4S={0, 10, 1100, 1101, 1110, 1111}→ ν(S4)={1, 2, 4, 4, 4, 4}.

Зауважимо також, що нерівність Крафта справджується не тільки для префіксних кодів, але й для будь-яких інших кодів, що однозначно декодуються.

Отже, кращим кодом був би код, для якого середня довжина кодових слів була б мінімальною. Для побудови такого коду потрібно знайти вектор Крафта при заданому розподілу ймовірностей.

Алгоритм Хафмена дає нам ефективний спосіб пошуку оптимального вектора Крафта при заданому ймовірнісному розподілі, для якого середня довжина кодових слів є мінімальною. Код, отриманий з використанням оптимального вектора Крафта при заданому розподілі ймовірностей появи символів, називають кодом Хафмена.

Алгоритм Хафмена реалізує загальну ідею статистичного кодування з використанням префіксних множин і працює у такий спосіб (код будується за допомогою бінарного дерева):

1. Розміщуємо всі символи алфавіту в порядку спадання ймовірностей їх появи, які приписуємо листям кодового дерева.

2. Послідовно об’єднуємо два символи з найменшими ймовірностями їх появи у новий вузол, ймовірність появи якого дорівнює сумі ймовірностей об’єднаних символів. Величина, що приписується вузлу дерева, називається його вагою. Два листки або вузли з найменшими значеннями ваги утворюють батьківський вузол, вага якого дорівнює сумарній вазі вузлів, що його створили. Надалі цей вузол враховується нарівні з вузлами і листками, що залишилися, а листя або вузли, що його створили, більше не розглядаються. Завершується процес побудовою кореня дерева з ймовірністю, що дорівнює 1.

3. Після побудови кореня кожна визначена гілка, що виходить з батьківського вузла, позначається 0 (як правило, якщо це ліва гілка) або 1 (права гілка). Кодові слова – це послідовності 0 і 1, що утворюються, на шляху від кореня кодового дерева до листків із задано ю імовірністю.

Деяка довільність у побудові дерева дозволяє отримати різні коди Хафмена з однаковою середньо ю довжиною коду. Виникає питання: «Який код Хафмена, побудований для даного алфавіту є найкращим»? Відповідь буде простою, хоча і не очевидною: кращим буде код з найменшою дисперсією. Дисперсія показує, наскільки сильно відхиляються довжини окремих кодів від їхньої середньої величини. Код Хаффмена є оптимальним ефективним кодом, тобто таким, що для джерела із заданими потужністю алфавіту та розподілом

ймовірностей появи символів гарантує найменшу середню довжину кодової комбінації.

Алгоритм Хафмена дає ефективний спосіб пошуку оптимального вектора Крафта при заданому ймовірнісному розподілі, для якого середня довжина кодових слів є мінімальною. Код, отриманий з використанням оптимального вектора Крафта при заданому розподілі ймовірностей появи символів, називають кодом Хафмена.

Нехай треба побудувати нерівномірний ефективний код для дискретного джерела, алфавіт якого має 8 символів А, Б,…, Ж, , ймовірності появи яких задано таблицею.

А	Б	В	Г	Д	Е	Є	Ж
0,2	0,6	0,1	0,04	0,015	0,025	0,01	0,01

Процес побудови кодового дерева є таким. Розміщуємо всі символи алфавіту в порядку спадання ймовірностей їх появи, тобто Б, А, В, Г, Е, Д, Є, Ж, які приписуємо листям кодового дерева. Далі вибираємо два вузли, яким відповідають найменші значення ймовірностей (це листя дерева Ж та Є), об’єднуємо їх, у результаті чого отримуємо новий вузол, якому приписуємо ймовірність, що дорівнює сумі ймовірностей об’єднаних вузлів – 0.02. Знову вибираємо два вузли з найменшими ймовірностями, але тепер не враховуємо вузли, що були об’єднанні (тобто Ж та Є), а беремо до уваги вузол, який з’явився (вузол з приписаною імовірністю 0.02) та вузол (листок) Д. Об’єднуємо їх у вузол та приписуємо ймовірність 0,035 (0.02+0.015). Повторюємо процедуру об’єднання вузлів з найменшими ймовірностями доки не утвориться кореневий вузол. Таким чином отримуємо кодове дерево. Тепер, використовуючи побудоване кодове дерево, запишемо кодові комбінації для символів джерела як послідовності символів 0 та1, що зустрічаються на шляху від кореня дерева до кінцевих вузлів, які зіставлені відповідним символам джерела (ліва гілка, що виходить з батьківського вузла, позначається 0, а права– 1). Результати побудови коду занесені в таблицю.

x_i	p_i	код
Б	0,6	0
А	0,2	11
В	0,1	101
Г	0,04	1001
Е	0,025	10001
Д	0,015	100001
Є	0,01	1000001
Ж	0,01	10000000

Середня довжина кодової комбінації $l = \sum p_i l_i = 1.815$ біт.

Контрольні питання

1. Що таке економне кодування інформації? З якою метою його застосовують?
2. Які є способи задання кодів?
3. Що таке рівномірні й нерівномірні коди?
4. Що таке надлишковість коду? Як її визначають?
5. З якою метою використовують оптимальні нерівномірні коди?
6. Які коди називають префіксними? Що таке вектор Крафта? Як записують нерівність Крафта?
7. Який критерій існування префіксного коду? У чому він полягає?
8. У чому полягає умова оптимальності префіксних кодів?
9. Що визначає пряма теорема посимвольного нерівномірного кодування?

ТЕМА 4. СЛОВНИКОВІ МЕТОДИ СТИСНЕННЯ

Далі йтиметься про алгоритми, а точніше - про деякі модифікації та вдосконалення двох алгоритмів, які є основою таких відомих комп'ютерних програм-архіваторів, як PKZIP, WinZip, LHA, ARJ, RAR та багатьох інших. їх розробили двоє ізраїльських учених Яків Зів та Абрахам Лемпел. У 1970-х роках вони опублікували алгоритми **LZ77** та **LZ78,** які започаткували нову еру в стисканні інформації. Практично всі сучасні комп'ютерні програми-архіватори використовують ту чи іншу версію алгоритму **LZ.**

Основною причиною популярності ***LZ-алгоритмів*** стала їхня простота та високий ступінь стиснення даних, а основна ідея полягає в тому, що другу і наступну появу деякої фрази (послідовності символів або одного символа) у повідомленні замінюють деяким вказівником на її попередню появу в повідомленні.

Крім простоти, ще однією важливою їхньою перевагою порівняно зі статистичними алгоритмами стиснення є швидка робота декодера. Це випливає з того, що декодування стисненого повідомлення відбувається простою заміною вказівника готовою фразою зі створеного словника, на яку він вказує.

Оскільки, ці алгоритми працюють з тим чи іншим виглядом словника, то їх назвали *словниковими алгоритмами стиснення.*

Усі алгоритми сім'ї **LZ** є універсальними алгоритмами стиснення інформації, у яких словник формується на підставі вже прочитаної та обробленої частини вхідного повідомлення, тобто адаптивно. Принципова відмінність - лише спосіб формування фраз (послідовності символів). У модифікаціях базових алгоритмів **LZ77** та **LZ78** ця властивість (відмінність) зберігається. Тому словникові алгоритми Лемпела-Зіва поділяють на дві сім'ї.

До першої сім'ї належать алгоритми, у яких весь словник створюється адаптивно (тобто в процесі стиснення) і не зберігається, а в неявному вигляді міститься у закодованих даних. Зберігаються лише неповторювані символи та

вказівники на повторювані фрази, що трапляються у повідомленні. Усі алгоритми цієї сім'ї ґрунтуються на алгоритмі **LZ77**.

Далі розглянемо базовий алгоритм **LZ77** та один з найдосконаліших представників цієї сім'ї - алгоритм **LZ78**, розроблений та опублікований 1982р. Й. Сторером і Т. Шиманським.

До другої сім'ї належать алгоритми, які початковий словник дікерела доповнюють новими фразами, що є повторюваними у повідомленні комбінаціями символів початкового словника та зберігають цей доповнений словник поряд з індексом фрази або окремого символа, що є кодом стисненого повідомлення. Усі алгоритми цієї сім'ї ґрунтуються на алгоритмі LZ78.

Базовий алгоритм LZ78 та один з найбільш досконалих представників цієї сім'ї - алгоритм **LZW,** розроблений та опублікований 1984 році Т.Уелчем.

Алгоритм LZ77

Як уже зазначено, алгоритм LZ77 є базовим для цілої сім'ї словникових алгоритмів, які ще називають алгоритмами з *ковзним словником* або *ковзним вікном.* Алгоритм LZ77 використовує вже прочитану й оброблену частину повідомлення як словник. У процесі кодування положення словника (вікна) відносно початку повідомлення постійно змінюється, словник наче ковзає по вхідному повідомленню. Щоб досягти стиснення, він намагається замінити черговий фрагмент повідомлення (зазвичай, найдовшу фразу) вказівником на таку ж фразу зі словника.

Ковзне вікно має довжину *N,* тобто у нього вміщується *N* символів, і розбите на дві нерівномірні частини. Перша, більша за розміром, містить уже прочитану частину повідомлення, її називають словником. Другу, набагато меншу, називають *буфером* або *випереджувальним буфером,* який містить частину ще не закодованого повідомлення. Зазвичай, розмір словника *M = M-F* становить декілька кілобайтів, а розмір буфера *F* набагато менший - до 100 байтів.

Кодування

Під час кодування вмісту буфера серед попередніх N-F символів, тобто

у словнику, відбувається пошук найдовшої фрази (послідовність символів), що збігається з фразою, яка є на початку буфера. Знайдений найбільший збіг кодує тріада *(i,j,s), де i* ~ зсув у словнику фрази, що збігається з фразою на початку буфера; j - довжина фрази, що збігається; *s*- перший символ, який є після фрази**,** що збігається, в буфері. Далі алгоритм зсовує увесь вміст вікна на j + **1** символів і водночас зчитує стільки ж символів вхідного потоку у буфер.

Зазначимо: якщо на деякому кроці збігу (деякої фрази чи бодай одного символа, що міститься на початку буфера) не виявлено, то алгоритм видає код <0,0,перший символ у буфері> < та продовжує роботу.

Обсяг пам'яті, що її потребує алгоритм-кодер або декодер, визначений розміром вікна ***N.*** Довжину коду обчислюють так: довжина фрази, що збіглася з вмістом словника, не може бути більшою від розміру буфера ***F,*** а зсув цієї фрази у словнику не може бути більшим від розміру словника мінус 1. Отже, довжина двійкового коду зсуву ***i*** буде $\log_2$(N-F)> довжина коду довжини фрази j - ("$\log_2$(F+1)"1, а символ s кодується 8 бітами за таблицею ASCII+.

Розглянемо декілька прикладів стиснення повідомлень алгоритмом LZ77. Процес стиснення інформації дуже зручно навести у вигляді таблиці, де вікно є нерухомим, а зміщується текст звідомлення.

Приклад 4.1. Стиснемо за алгоритмом LZ77 повідомлення **"А-БА-БА-ГА-ЛА-МА-ГА";** розмір словника – 8 байтів, буфера – 5 байтів.

Кодування повідомлення наведено в таблиці.

Таблиця 4.1 – до прикладу 4.1. Стиснення повідомлення за алгоритмом LZ77

Словник (8 байт)								Буфер (5 байт)					Код
0	1	2	**3**	4	5	6	7	1	2	3	4	5	
								А	**-**	**Б**	**А**	**-**	**<0,0,'А'>**
							А	**-**	**Б**	**А**	**-**	**Б**	**<0,0,'-'>**
						А	**-**	**Б**	**А**	**-**	**Б**	**А**	**<0,0,'Б'>**
					А	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**<5,3,'А'>**
	А	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**г**	**А**	**-**	**Л**	**<5,1,'1'>**
-	**Б**	**А**	**-**	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**<5,2,'Л'>**
-	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**м**	**А**	**-**	**<5,2,'М'>**
-	**Г**	**А**	**-**	**Л**	**А**	**-**	**м**	**А**	**-**	**г**	**А**		**<5,2,'Г'>**
-	**Л**	**А**	**-**	**м**	**А**	**-**	**г**	**А**					**<0,0,'А'>**

В останньому рядку таблиці буква **А** взята не зі словника, оскільки вона

остання.

Довжина отриманого коду L=9-(3+3+8) = 126 біт

проти L = 19-8 = 152 біт коду нестисненого рядка.

Приклад 4.2. Стиснемо за алгоритмом LZ77 повідомлення **“АБРАКАДАБРА”;** розмір словника - 8 байтів, буфера - 5 байтів.

Кодування повідомлення наведено в таблиці 4.2.

Таблиця 4.2 – до прикладу 4.2. Стиснення повідомлення за алгоритмом LZ77

Словник (8 байт)								Буфер (5 байт)					Код
0	1	2	3	4	5	6	7	1	2	3	4	5	
								А	**Б**	**Р**	**А**	**К**	**<0,0,’А’>**
							А	**Б**	**Р**	**А**	***К***	**А**	**<0,0,’Б’>**
						А	**Б**	**Р**	**А**	**К**	**А**	**д**	**<0,0,’Р’>**
					А	**Б**	**Р**	**А**	**К**	**А**	**д**	**А**	**<5,1,’К’>**
			А	**Б**	**Р**	**А**	**К**	**А**	**д**	**А**	**Б’**	**Р**	**<3Д,2Д’>**
-	**А**	**Б**	**Р**	**А**	**К**	**А**	**д**	**А**	**Б**	**Р**	**А**		**<1,2,’Р’>**
р	**А**	**К**	**А**	**д**	**А**	**Б**	**р**	**А**					**<0,0,’А’>**

Довжина отриманого коду L= 7-(3+3+8) = 98 біт проти L= 88 біт коду нестисненого рядка.

Цей приклад засвідчує, що кодування поодиноких символів за алгоритмом LZ77 є неефективним, оскільки збільшує загальну довжину коду, тобто стиснення інформації не відбувається.

Декодування

Під час декодування виконується той самий порядок роботи з вікном, що й під час кодування, однак, на відміну від пошуку підрядків, що співпадають, їх, навпаки, копіює декодер з вікна згідно з черговою тріадою коду.

Приклад 4.3. Декодуємо повідомлення, закодоване за алгоритмом LZ77, довжина словника - 8 байтів. Код стисненого повідомлення:

<0,0,’А’ ><0,0,’-’ ><0,0,’Б’><5,3,’А’><5,1,Т’>
<5Д,’Л’ ><5,2,’М’ ><5,2,Т’><0,0,’А’>.

Декодування повідомлення наведено нижче.

Таблиця 4.3 – декодування повідомлення за алгоритмом LZ77

Вхідний код	Вихідний символ	Словник (8 байт)							
		0	1	2	3	4	5	6	7
<0,0,'А'>	**А**								**А**
<0,0,'-'>	**-**							**А**	**-**
<0,0,'Б'>	**Б**						**А**	**-**	**Б**
<5Д'А'>	**А-БА**		**А**	**-**	**Б**	**А**		**Б**	**А**
<5,1, 'Г'>	**-Г**	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**Г**
<5,2,'Л'>	**А-Л**	**-**	**Б**	**А**		**Г**	**А**		**Л**
5 <5,2,'М'>	**А-М'**		**Г**	**А**	**-**	**Л**	**А**	**-**	**М**
Р <5Д,'Г'>	**А-Г**		**Л**	**А**		**М**	**А**		**Г**
У «0,0,'А'>	**А**	**Л**	**А**		**М**	**А**	**-**	**Г**	**А**

Проблеми LZ77

Очевидно, що швидкодія кодера LZ77 значно залежить від того, як відбуватиметься пошук однакових фраз у словнику. Якщо шукати збіг повним перебиранням можливих варіантів, то очевидно, що стискання буде дуже повільним. Причому зі збільшенням розміру вікна для підвищення ступеня стиснення швидкість роботи буде пропорційно сповільнюватися. Для декодера це не важливо, оскільки він не виконує пошуку на збіжність.

Швидкодія кодера залежить від того, як організовано ковзання вікна по повідомленню. Один зі способів роботи з вікном - користуватися кільцевим буфером. Хоча для підтримки кільцевого буфера необхідні додаткові витрати, загалом це дає суттєву вигоду, бо нема постійного зсуву великого блоку пам'яті.

Крім проблем зі швидкодією, в алгоритму LZ77 виникають проблеми з самим стисканням інформації. Вони з'являються тоді, коли кодер не може знайти тотожної фрази у словнику і видає стандартну тріаду, кодуючи один символ. У цьому випадку, наприклад, якщо словник має довжину 4К, а буфер - 16 байтів, то код, який видає кодер <0,0, символ>, займатиме 3 байти (12 біт - для величини зсуву, 4 - для довжини однакових символів, 8 біт - для символа). Отже, один байт кодується трьома. Це дуже неефективно і призводить до збільшення розміру файлу, а не до стиснення.

Отже, недоліки алгоритму LZ77 полягають у такому: 1) зі збільшенням розміру словника швидкість роботи алгоритму кодера пропорційно сповільнюється; 2) кодування поодиноких символів дуже неефективне.

Алгоритм LZ88

Алгоритм LZ88 є модифікацією алгоритму LZ77. Його розробники

намагалися уникнути тих недоліків, які є в його попередника. Алгоритм LZ88 відрізняється від алгоритму LZ77 тим, як підтримується ковзне вікно, та кодами кодера.

Кодування

Код алгоритму починається однобітовим префіксом, який відділяє код фрази, що збігається, від незакодованого символа. Код складається з пари **<i,** _j> - зсуву і у словнику фрази, що збігається з фразою на початку буфера, і довжини j цієї фрази**,** і для алгоритму LZ77. Вікно зсувається рівно на довжину знайденого підрядка або на 1, якщо входження підрядка буфера у ‘словнику не знайдено. Такі коди дають змогу отримати суттєву перевагу над алгоритмом LZ77 у розмірах стисненого повідомлення.

Довжина фрази у алгоритмі LZ88 завжди більше **0** і не може перевищувати розмір буфера **F,** тому довжина двійкового коду довжини фрази, що збігається, j буде $[\log_2 F]$, а довжина коду $i\text{-}\log_2(N\text{-}F)$.

Для порівняння розглянемо роботу алгоритму LZ78 на тих же прикладах, що й алгоритму LZ77.

Приклад 4.3. Стиснемо за алгоритмом LZ78 повідомлення **“А-БА-БА-ГА-ЛА-МА-ГА”**; розмір словника - 8 байтів, буфера - 5 байтів.

Кодування повідомлення наведене в таблиці. Довжина отриманого коду L= 8(1+8)+5(1+3+3) = 72+35=107 (біт)

проти 152 біт коду нестисненого рядка та L= 9(3+3+8) = 126 біт коду за алгоритмом LZ77.

Таблиця 4.4 – до прикладу 4.3. Стиснення повідомлення за алгоритмом LZ78

Словник (8 байт)								Буфер (5 байт)					Код
0	1	2	3	4	5	*6*	7	1	2	3	4	5	
								А	**-**	**Б**	**А**	**-**	**О’А’**
							А	**-**	**Б**	**А**	**-**	**Б**	**0’-’**
						А	**-**	**Б**	**А**	**-**	**Б**	**А**	**О’Б’**
					А	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**I<5,3>**
		А	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**Г**	**А**	**-**	**1<5,2>**
А	**-**	**Б**	**А**	**-**	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**ОТ’**
-	**Б**	**А**	**-**	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**1<5,2’>**
А	**-**	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**М**	**А**	**О’Л’**
-	**Б**	**А**	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**М**	**А**	**-**	**1<5,2>**
А	**-**	**Г**	**А**	**-**	**Л**	**А**	**-**	**М**	**А**	**-**	**Г**	**А**	**О’М’**
-	**Г**	**А**	**-**	**Л**	**А**	**-**	**М**	**А**	**-**	**Г**	**А**		**1<5,2>**
А	**-**	**Л**	**А**	**-**	**М**	**А**	**-**	**Г**	**А**				**ОТ’**
-	**Л**	**А**	**-**	**М**	**А**	**-**	**Г**	**А**					**О’А’**

Приклад 4.4. Стиснемо за алгоритмом LZ78 повідомлення **“АБРАКАДАБРА”**; розмір словника - 8 байтів, буфера - 5 байтів.

Кодування повідомлення наведено в таблиці.

Таблиця 4.5 – до прикладу 4.4. Стиснення повідомлення за алгоритмом LZ78

Словник (8 байт)								Буфер (5 байт)					Код
0	1	2	3	4	5	6	7	1	2	3	4	5	
								А	**Б**	**Р**	**А**	**К**	**О’А’**
							А	**Б**	**Р**	**А**	**К**	**А**	**О’Б’**
						А	**Б**	**Р**	**А**	**К**	**А**	**Д**	**О’Р’**
					А	**Б**	**Р**	**А**	**К**	**А**	**Д**	**А**	**1<5,1>**
				А	**Б**	**Р**	**А**	**К**	**А**	**Д**	**А**	**Б**	**О’К’**
			А	**Б**	**Р**	**А**	***К***	**А**	**Д**	**А**	**Б**	**Р**	**1<5,1>**
		А	**Б**	**Р**	**А**	**К**	***А***	**Д**	**А**	**Б**	**Р**	**А**	**О’Д’**
	А	**Б**	**Р**	**А**	**К**	**А**	**Д**	**А**	**Б**	**Р**	**А**		**1<13>**
Р	**А**	**К**	**А**	**Д**	**А**	**Б**	**Р**	**А**					**О’А’**

Довжина отриманого коду L =6-(1+8)+3(1+3+3) = = 54+21 = 75 біт проти L= 7 (3+3+8) = 98 біт коду за алгоритмом *LZ77* та $L= 11\text{-}8 = 88$ біт коду нестисненого рядка.

Декодування

Алгоритм LZ78, узагалі, є дуже асиметричним за часом. Якщо процедура стискання достатньо складна і виконує великий обсяг роботи під час обробки кожного символа, то декодер алгоритму LZ78 є тривіально простим і може працювати зі швидкістю, яка наближається до швидкості процедури звичайного копіювання інформації.

Отже, декодер читає один біт стисненої інформації (префікс коду) і визначає, чи це символ, чи пара <зсув, довжина> (0 - це символ, 1 - пара). Якщо це символ, то наступні 8 бітів видаються як розкодований символ і заносяться у ковзне вікно. Якщо це пара, то відповідна кількість символів словника заноситься у ковзне вікно та видається у розкодованому вигляді. Оскільки це все, що робить декодер, то зрозуміло, чому він працює так швидко.

Приклад 4.5. Декодуємо повідомлення, закодоване за алгоритмом LZ78, довжина словника 8 байтів. Код стисненого повідомлення: 0’А’0’Б’0’Р’1<5,1>0’К’1<5,1>0’Д’1<1,3 >0’А’.

Декодування повідомлення наведено в таблиці.

Таблиця 4.6 – декодування повідомлення за алгоритмом LZ78

Вхідний код	Вихідний **СИМВОЛ**	Словник (8 байт)							
		0	1	2	3	**4**	5	6	7
О'А'	**А**								**А**
О'Б'	**Б**							**А**	**Б**
О'Р'	**Р**						**А**	**Б**	**Р**
1<5,1>	**А**					**А**	**Б**	**Р**	**А**
О'К'	**К**				**А**	**Б**	**Р**	**А**	**К**
1<5,1>	**А**			**А**	**Б**	**Р**	**А**	**К**	**А**
О'Д'	**Д**		**А**	**Б**	**Р**	**А**	**К**	**А**	**д**
1<13>	**АБР**	**Р**	**А**	**К**	**А**	**д**	**А**	**Б**	**р**
О'А'	**А**	**А**	**К**	**А**		**А**	**Б**	**Р**	**А**

Алгоритми LZ77, LZ78 мають суттєвий недолік. Оскільки вони як словник використовують лише невеликий фрагмент повідомлення, то це не дає змоги кодувати повторювані фрази, відстань між якими у повідомленні є більшою, ніж розмір словника. Крім того, алгоритми обмежують розмір фрази (розміром буфера), яку можна закодувати. Наприклад, якщо словник має довжину 4К, а буфер - 16 байтів, то розмір фрази обмежений 16 байтами. Очевидно, що наведені чинники знижують ефективність кодування.

На перший погляд, виходом з цієї ситуації є збільшення розмірів буфера, однак механічне збільшення розмірів словника та буфера або, зокрема, тільки буфера, зазвичай, дає протилежні до очікуваних результати. Припустимо, що ми збільшимо розмір словника з 4К до 64К, а розмір буфера з 16 байт до 1К. Це приведе до того, що для кодування зсуву у словнику потрібно буде 16 біт замість 12, а для кодування довжини фрази - 10 біт замість 4. Отже, кожна фраза буде закодована 26 бітами замість 16, що зробить загальний код повідомлень з короткими повторюваними фразами неприпустимо великим.

Крім того, різко зросте час роботи алгоритму кодера. Наприклад, збільшення розмірів словника з 4 до 64К в алгоритмі LZ77 приведе до зростання в 16 разів часу пошуку найдовшого збігу, а зі збільшенням розміру буфера з 16 байт до 1К ця операція виконуватиметься в 64 рази довше для обох алгоритмів, оскільки як в алгоритмі LZ77, так і в алгоритмі LZ88 відбувається посимвольне порівняння вмісту буфера з фразою у словнику.

Отже, теоретично потужніші (унаслідок збільшення розмірів словника

та буфера) алгоритми стають практично непридатними.

Алгоритм LZW — це найпопулярніший варіант алгоритму LZ78, розроблений Т. Уелчем у 1984 р. Він відрізняється від LZ78 приблизно тим, чим LZ88 відрізняється від LZ77, а саме: довжина коду зменшується на одне (друге) поле, тобто кодер ніколи не видає самі символи повідомлення, яке стискається, а видає лише вказівники на місце символа або фрази у словнику.

Кодування

Робота алгоритму LZW починається ініціалізацією словника, наприклад, розміром 4К всіма символами вхідного алфавіту (таблиця A8CII+) за адресами від 0 до 255. А адреси від 256 до 4095 заповнюватимуться фразами у процесі стискання.

Оскільки словник уже частково заповнений, то перші зчитані символи повідомлення завжди будуть знайдені у словнику. Тому коди складаються тільки з вказівника на символ або фразу у словнику (адреса символа або фрази у словнику) і мають фіксовану довжину, що визначена величиною $|\log_2([\text{розмір словника}])]$ (у нашому випадку 12 біт).

У процесі кодування алгоритм додає символи, які зчитуються, до накопичувального рядка і після кожного нового доданого символа, кодер шукає його у словнику. Якщо рядок знайдено, то процес продовжується. Як тільки додавання нового символа S приводить до того, що рядка wS у словнику немає, тоді кодер:

1) записує у вихідний файл вказівник, під яким рядок міститься у словнику;
2) зберігає накопичувальний рядок (який тепер називатимемо фразою) у словнику на наступній допустимій позиції (присвоює їй наступний вільний вказівник);
3) присвоює (ініціалізує) накопичувальному рядку н нове значення.

Далі розглянемо деякі специфічні процедури стиснення графічних файлів, які враховують особливості представлення зображень при виведенні їх на екран монітора або інформаційному обміні в телекомунікаційних системах.

Контрольні питання

1. Яка мета стиснення інформації? Які основні елементи охоплює система стиснення інформації? Які функції вони виконують?
2. За якими ознаками класифікують системи й методи стиснення даних?
3. Чим відрізняються системи стиснення інформації, призначені для передавання й архівації даних?
4. У чому полягають словникові методи стиснення LZ78, LZW?
5. Чим визначена довжина кодів для алгоритмів LZ78, LZW? Які переваги модифікації LZW?
6. Які переваги алгоритмів LZ78, LZW порівняно з LZ77, LZSS?

ТЕМА 5. АЛГОРИТМИ СТИСКАННЯ ЗОБРАЖЕНЬ

Алгоритми *RLE*

Кодування довжин повторів *(Run Length Encoding* (RLE)) - загальна назва методів стиснення даних, зокрема архівування зображень (графіки), без втрат, при якому серії однакових символів замінюються простою структурою, в якій вказується код даних та коефіцієнт повторення. Зазвичай, метод RLE є одним з кроків багатокрокового алгоритму стиснення, у якому додатково проводиться статистичне або словникове стиснення.

Приклад. Нехай потрібно закодувати такі дані

X = АААААББББББББББВВАДДДДКК5

довжиною 25 байт. Унаслідок роботи алгоритму отримуємо кодове слово

Code(X) = 5 А1ОБВВ А4 ДКК5

довжиною 12 байт.

Як бачимо з прикладу, послідовності (ланцюжки) з п’яти символів “А”, десяти символів “Б” та чотирьох символів “Д” кодуються двома символами “5” та “А”, “10” та “Б” і “4” та “Д”, відповідно. Послідовності “ВВ”, “КК” та окремі символи “А” та “5” не перекодовуються, оскільки немає виграшу від заміни таких об’єктів на послідовності типу “довжина”+“символ”.

У разі реалізації стиснення за цим методом виникають труднощі такого характеру, як декодер відрізнить інформацію про довжину ланцюжка однакових символів від нестиснених даних? Зокрема, наприклад, як декодер відрізнить керівний символ “5” перед символом “А” від символа даних з таким самим кодом (останній символ - “5”)?

Є декілька варіантів вирішення цієї проблеми.

Перший варіант алгоритму RLE

В основі першого варіанта алгоритму RLE є ідея виявлення послідовностей даних, що повторюються, та заміни цих послідовностей структурою, яка складається з керівного символа, коду даних і коефіцієнта

повторення. Для зручності керівний символ називають префіксом. За префікс беруть символ, який найменше трапляється в даних, що стискаються або не трапляється взагалі. Він записується на початку стисненого файлу для того, щоб декодер знав, який байт є префіксом.

Тепер послідовність “AAAAA” з прикладу кодером буде закодована у трійку: префікс (8 біт), символ “A” (8 біт), коефіцієнт повторення “5”, тобто 5 байт будуть закодовані трьома. Якщо у вхідному файлі під час стискання кодер виявить байт з кодом префікса, то у вихідний файл запише два байти з кодом префікса і за цією ознакою декодер визначить, де префікс є символом даних, а де - символом керування.

Отже, у підсумку роботи алгоритму отримуємо кодове слово Code(X)=ННА5НБ10ВВАНД4КК5.

Коефіцієнт стиснення ***r*** w **1.5625.**

У кодовому слові перший байт “Н”‘ - це байт префікса, щоб декодер знав, який байт є префіксом.

Можливі модифікації цього варіанта, наприклад, кодувати послідовності символів та одиничні символи двома байтами типу “довжина” + “символ”, де префіксом повторюваності слугують одиниці у двох старших бітах байту “довжини”, тобто на саму довжину ланцюжка виділяється 6 біт, довжина може набувати значення від 1 до 63 двійкових символів, тобто накладається обмеження на кількість повторюваних символів у ланцюжку. Якщо у ланцюжку більше, ніж 63 символи, то кодуються ланцюжки по 63 символи плюс остача. Отже, у цьому випадку послідовність з 63 байт однакових символів стискається у 2 байти, тобто у 32 рази.

Такий варіант алгоритму розрахований на ділову графіку - зображення з великими ділянками кольору, що повторюється. Він реалізований у форматі PCX**.**

Другий варіант алгоритму RLE

Ідея другого варіанта алгоритму полягає в тому, щоб не кодувати кожний одиничний символ двома байтами типу “довжина” + “символ”, а

визначати кількість символів, які не перекодовуються під час стиснення у байті “довжина”, тобто кількість символів, які кодер пропускає.

Отже, префіксом повторюваності, як і у попередньому алгоритмі, слугує одна одиниця у старшому біті байта “довжини”, а префікс 0 (0 у старшому біті байту “довжини”) означає, що за байтом “довжина” йде відповідна кількість (визначена у байті “довжина”) неповторюваних символів, тобто у цьому варіанті алгоритму на довжину також накладається обмеження - 7 біт, тобто можуть кодуватися ланцюжки довжиною не більше 127 символів, а також стільки ж символів пропускатися, тобто декодер, виявивши префікс 0, зрозуміє, що наступні *k* (1 ... 127**)** символів не були стиснуті кодером.

Отже, такий варіант алгоритму має більший максимальний ступінь стиснення, ніж перший варіант алгоритму, завдяки тому, що послідовність уже з 127 байт стискається у 2 байти, тобто у 64 рази і, крім того, якщо трапляється послідовність неповторюваних символів, то двома байтами кодується не кожний символ, а вся послідовність неповторюваних символів.

Такий варіант алгоритму реалізований у форматах TIFF та TGA.

Третій варіант алгоритму RLE

Цей варіант алгоритму кодування довжин повторень використовують для стиснення таких цифрових даних, у яких є ділянки з великою кількістю нульових даних.

Ідея алгоритму полягає в тому, що кожного разу, коли серед цифрових даних трапляється “0”, то він кодується двома числами. Перше - число 0, яке є ознакою початку кодування довжини ланцюжка нулів, друге - кількість нулів у ланцюжку. Якщо середня кількість нулів у всіх ланцюжках, включаючи ланцюжки з одного нуля (поодинокі нулі), є більше двох, то стиснення інформації відбудеться. Якщо ж у даних наявна велика кількість поодиноких нулів, то цей варіант алгоритму є неефективним і може призвести навіть до збільшення розміру файлу.

Приклад. Нехай треба закодувати такі цифрові дані:

X=5 *7* 55 8 71 0000 99 5 13 031 23 000003 67 007...

довжиною 25 байт. Унаслідок роботи алгоритму отримаємо кодове слово

Code(X) = ***5*** 7 55 8 71 0 4 99 5 13 0 1 31 23 0 5 3 67 0 2 7

довжиною 21 байт. Ефективність (коефіцієнт) стиснення r= 1.19, тобто стиснення даних у цьому випадку відбувається, оскільки середня кількість нулів у всіх ланцюжках дорівнює 3 (>2). Для наочності (і тільки) усі ланцюжки з нулів у вхідній послідовності виділено жирним шрифтом. У кодовому слові, знову ж таки тільки для наочності, виділено ту пару чисел, якою кодується кожний ланцюжок.

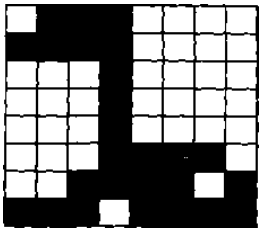

Рисунок 5.1 – Фрагмент двійкового зображення

Проскануємо це зображення по рядках (двом кольорам на зображенні, наприклад, чорному та білому, відповідатимуть двійкові символи **0** та 1). У підсумку отримаємо двійковий вектор даних

X=0111000011110000000100000001000000010000000011110001111011110111 1

довжиною 64 біт.

Виділимо у векторі *X* послідовності (ланцюжки) з однаковими даними (це будуть ланцюжки лише з нулів та одиниць) та визначимо їхні довжини. Сумарна послідовність довжин ланцюжків (додатних цілих чисел), яка відповідає вектору *X*, матиме вигляд

У(X) =13447171717434141 4.

Тепер цю послідовність можна закодувати одним зі статистичних алгоритмів, наприклад, класичним алгоритмом Хафмена.

Для того щоб знати, що закодована послідовність починається з 0, на початку кодового слова записуємо префіксний символ 0. У результаті отримуємо кодове слово

Code(У(X)) = 0101110011010110101101011001110100100

Приклад. Нехай потрібно закодувати цифрові дані, серед яких є поодинокі нулі, наприклад:

X = ***5*** 7 0 0 5508 71 0995 13 031 23 03 6700704509...довжиною 25 байт. Унаслідок роботи алгоритму отримаємо кодове слово

Code(X) = 5 702 55 018 71 01 99 5 13 0131 23 013 67 027014501 9 довжиною 31 байт. Ефективність (коефіцієнт) стиснення 0.81, тобто не тільки не відбувається стиснення даних, а й кодування за цим варіантом алгоритму призводить до збільшення розміру файлу.

Четвертий варіант алгоритму RLE

Цей варіант алгоритму застосовують для стиснення двійкового зображення. Ідея такого алгоритму на підставі кодування довжин повторень та одного зі статистичних алгоритмів полягає в тому, що виявляються послідовності (ланцюжки) даних, які складаються з однакових елементів та кодуються за допомогою одного зі статистичних алгоритмів лише довжини цих ланцюжків.

Оскільки ланцюжки складаються лише з нулів та одиниць, то у вихідній послідовності довжин цих ланцюжків вони (довжини) строго перемежовуватимуться. Тому немає потреби визначати код даних для кожного ланцюжка, як у перших двох алгоритмів, а лише на початку кодового слова задати, з якого символа - 0 чи 1 - починається закодована послідовність.

Отже, кодове слово складається з префіксного символа (1 біт) та послідовності довжин ланцюжків, які строго чергуються між собою.

У разі стискання зображень великого розміру, які містять багато елементів, що повторюються, ефективність стиснення суттєво зростає.

Алгоритм диференціального кодування

Для багатьох типів даних різниця між сусідніми відліками (цифровими значеннями) даних порівняно невелика, навіть якщо самі дані є великими значеннями. Наприклад, немає великої різниці між сусідніми пікселами цифрового зображення. Ідея алгоритму диференціального кодування полягає в тому, що кодуються різниці між сусідніми відліками, а не самі відліки.

Наведений нижче простий приклад дає змогу зрозуміти, яку перевагу може дати диференціальне кодування порівняно з простим кодуванням без пам’яті, тобто кодуванням відліків незалежно один від одного рівномірним кодом.

Приклад**.** Нехай десять послідовних пікселів 8-бітового (256-рівневого) цифрового зображення мають такі рівні:

X= 144 147 150 146 141 142 138 143 145 142

Обчислимо у векторі *X* різниці між сусідніми пікселями. Отримана послідовність різниць (цілих чисел зі знаком) між сусідніми пікселями, яка відповідає вектору *X* , матиме вигляд

d(X)= 144 3 3 -4 -5 1 -4 5 2 -3.

Для кодування першого числа послідовності *d(X)* потрібно 8 біт, а всі решта числа можна закодувати 4-бітовими словами (один знаковий біт і три біти на кодування модуля числа).

Отже, унаслідок кодування отримаємо кодове слово довжиною 8+9x4=44 біт. Це майже вдвічі коротше, ніж у разі прямого кодування кожного відліку рівномірним 8-бітовим кодом (10x8 = 80 біт).

Під час декодування послідовність *X* легко відновлюється з різницевої послідовності *d(X)* простим підсумовуванням (дискретним інтегруванням)

Метод диференціального кодування дуже широко використовують у тих випадках, коли природа даних така, що їхні сусідні значення незначно відрізняються один від одного, при тому що самі значення можуть бути як завгодно великими.

Це стосується звукових сигналів, особливо мови, зображень, сусідні пікселі яких мають практично однакові яскравості і колір тощо. Водночас цей метод абсолютно не придатний для кодування текстів, креслень або будь-яких цифрових даних з незалежними сусідніми значеннями.

Контрольні питання

1. У чому полягає алгоритм кодування довжин повторів (RLE)?
2. Які є варіанти алгоритму RLE? Чим вони відрізняються?
3. У чому полягає алгоритм диференціального кодування?
4. Для яких даних диференціальне кодування є ефективнішим, ніж просте кодування, і чому?
5. Які є найпопулярніші моделі передавання кольору? Де їх використовують?

ТЕМА 6. КОЛЬОРОВІ МОДЕЛІ

Відомо, що кольорове зображення потребує не менше трьох чисел на один піксель для того, щоб точно передати його колір.

Метод, вибраний для відображення яскравості й кольору, називають *кольоровим простором,* або *кольоровою моделлю.*

Є три найпопулярніші кольорові моделі - RGB (яку використовують у комп'ютерній графіці), YIQ, YUV або YCbCr (які використовують у відеосистемах), та CMYK (яку використовують у кольоровому друці). Усі кольорові моделі можна отримати з моделі RGB.

RGB

Цю кольорову модель найбільше застосовують у комп'ютерній графіці. Червоний, зелений та блакитний - головні компоненти кольорів, які відображають три розмірності цього кольорового простору. Діагональ куба з однаковими значеннями RGB передає градації сірого кольору від чорного до білого.

Кольорові ЕПТ - електронно-променеві трубки та рідкокристалічні дисплеї відображають RGB зображення, окремо підсвічуючи червоні, зелені й блакитні компоненти кожного піксела. Якщо дивитися на екран з відстані звичайного глядача, то різні компоненти зливаються у єдиний "правильний колір".

RGB придатна для комп'ютерної графіки, оскільки для формування кольору там використовують саме ці три компоненти зображення. Річ у тому, що для збереження кольору зображень необхідно знати і зберігати всі три компоненти RGB, і втрата однієї з них значно спотворить візуально якість зображення. Також у разі обробки зображень у моделі RGB не завжди зручно змінювати тільки яскравість або контрастність окремого пікселя, оскільки в цьому випадку необхідно прочитати всі три значення компонент RGB, перерахувати їх для потрібної яскравості та записати назад. З цієї та інших причин багато стандартів відео використовують *яскравість* та *два кольоророзницеві сигнали* як кольорову модель, що відрізняється від RGB. Найбільш відомими серед таких кольорових моделей є YIQ, YUV та YCbCr. Незважаючи на те, що всі вони пов'язані між собою, між ними все ж таки є деякі відмінності.

Модель YCbCr

Відомо, що органи зору людини менш чутливі до кольору предметів, ніж до їхньої яскравості. У кольоровій моделі RGB усі три компоненти вважають однаково важливими, вони зазвичай зберігаються з однаковою роздільною здатністю. Однак можна відтворити кольорове зображення ефективніше, якщо відокремити яскравість (світність) від кольорової інформації, відобразивши її з більшою роздільною здатністю, ніж колір. Тому кольорова модель YCbCr та її варіації є популярним методом ефективного відтворення кольорових зображень.

Літера Y в таких кольорових просторах позначає компоненту яскравості, яку обчислюють як зважене усереднення компонент R, G і B за такою формулою

$$Y = k_r R + k_g G + k_b B,$$

де k позначає відповідний ваговий множник. Інші кольорові компоненти, по суті, визначають у вигляді різниці між яскравістю Y та компонентами R, G і B:

$$C_b = B - Y,$$
$$C_r = R - Y,$$
$$C_g = G - Y.$$

У цьому разі отримують чотири компоненти нової моделі замість трьох у RGB. Однак число $C_b + C_r + C_g$ є сталим, тому тільки дві з трьох хроматичних компонент необхідно зберігати, а третю обчислювати на їхній підставі. Найчастіше як дві кольорові компоненти використовують C_b та C_r. Переваги моделі YCbCr порівняно з RGB полягають у тому, що C_b та C_r можна відображати з меншою роздільною здатністю, ніж Y, оскільки око людини менш чутливе до кольору, ніж до його яскравості. Це дає змогу зменшити обсяг інформації, яка потрібна для відображення хроматичних компонент, без помітного *погіршення якості* передавання колірних відтінків зображення. Такий підхід до перетворення кольорового простору дає додатковий ефект у разі стискання зображень. У цьому випадку алгоритми стиснення спочатку перетворюють вихідну кольорову модель RGB у YCbCr, стискають інформацію, а потім під час відновлення, знову перетворюють зображення в кольорову модель RGB, оскільки її використовують у комп'ютерах. Формула для прямого перетворення має вигляд:

$$Y = k_r R + (1 - k_b - k_r) G + k_b B,$$

Рекомендація ITU-T (міжнародний стандарт у телекомунікації) пропонує такі коефіцієнти:

$$k_b=0{,}114,\ k_r=0{,}229.$$

Використовуючи ці значення в наведених рівняннях, отримаємо поширені формули

$$Y = 0{,}2995+ 0{,}587G +0{,}1145,\ C_b =0{,}564(B-Y),$$

$$C_г =0{,}713(R-Y).$$

$$R = Y + 1{,}402C_г,$$

$$G = У-0{,}344C^*-0{,}714C_г,$$

$$B = Y + 1{,}772C_b.$$

Хроматичні компоненти C_b та C_r можна відобразити з меншою роздільною здатністю, ніж компоненту яскравості Y. У цьому разі на практиці використовують такі формати їхнього взаємного відображення.

Найочевидніший формат - це так званий формат 4:4:4, який забезпечує цілковиту точність у передаванні хроматичних компонент, тобто на кожні чотири світлові відліки Y передаються по чотири відліки компонент C_b та C_r. Інший формат 4:2:2 (YUY2) передбачає, що на кожні чотири відліки компоненти *Y* припадає по два відліки хроматичних компонент. Такий формат використовують для високоякісного кольорового відео і в стандартах *MPEG-4* та H.264.

Найпопулярніший формат 4:2:0 . У ньому кожна компонента C_b та $C_г$ має один відлік на чотири відліки *Y*. Причому відліки компонент C_b та $C_г$, зазвичай, обчислюють двома способами. У першому випадку виконують інтерполяцію за найближчими чотирма відліками компонент C_b та C_r, щоб сформувати один відлік для них.

Такий підхід застосовують у стандартах *MPEG-1* та H.261, H.263. У іншому випадку виконують інтерполяцію за двома вертикальними відліками і застосовують у стандарті *MPEG-2*.

Завдяки економному зображенню кольорових сцен формат 4:2:0 широко використовують у багатьох додатках, таких як відеоконференції, цифрове телебачення, DVD. Оскільки хроматичні компоненти відбирають у 4 рази рідше, ніж компоненту яскравості, то простір 4:2:0 моделі YC_bC_r займає у два рази менше відліків порівняно з форматом відео 4:4:4 моделі RGB.

Алгоритм стискання JPEG

Алгоритм розроблений групою експертів у галузі фотографії (Joint

Photographic Expert Group) спеціально для стискання 24- бітових та півтонових зображень 1991 р. Цей алгоритм дуже добре стискає (обробляє) зображення з неперервними тонами, у яких близькі пікселі мають подібні кольори. Зазвичай око не в змозі помітити якоїсь різниці в разі стискання цим методом у 10 або 20 разів.

Ґрунтується алгоритм на дискретному косинус-перетворенні (ДКП), яке застосовують до матриці блоків зображення, які не перетинаються, розміром 8x8 пікселів. ДКП розкладає ці блоки за амплітудами деяких частот. У результаті отримують матрицю, у якій багато коефіцієнтів, зазвичай, близьких до нуля, які можна зобразити в грубій числовій формі, тобто у квантованому вигляді без суттєвої втрати якості під час відтворення.

Розглянемо роботу алгоритму детальніше. Припустимо, що стискають повне кольорове 24-бітове зображення. У цьому випадку маємо такі етапи роботи.

Крок 1. Переводимо зображення з моделі RGB у модель YCbCr.

Зазначимо, що обернене перетворення легко отримати множенням оберненої матриці на вектор

[Y, C_b, C_r]^T-[0, 128, 128]^T, який, по суті, є моделлю YUV:

Крок 2. Розбиваємо вихідне зображення на матриці 8x8. Формуємо з кожної три робочі матриці ДКП - по 8 біт окремо для кожної компоненти. У разі великих ступенів стискання блок 8x8 розкладається на компоненти YCbCr у форматі 4:2:0, тобто компоненти для C_b та C_r беруть через точку по рядках і стовпцях.

Крок 3. Застосовуємо ДКП до блоків зображення 8x8 пікселів. Оскільки ДКП є "серцем" алгоритму *JPEG,* то бажано на практиці обчислювати його якнайскоріше. Простим підходом для прискорення обчислень є завчасне обчислення функцій косинуса та зведення результатів у таблицю. Крім того, враховується ортогональність функцій косинусів з різними частотами.

Унаслідок ДКП отримуємо матрицю *Y, у* якій коефіцієнти в лівому куті відповідають низькочастотній складовій зображення, а в правому нижньому - високочастотній.

Крок 4. Квантування. На цьому кроці відбувається відкидання частини інформації. Тут кожне число з матриці *Y* ділиться на спеціальне число з *"таблиці квантування",* а результат округляється до найближчого цілого:

Причому для кожної матриці *Y,* C_b та C_r можна задавати свої таблиці

квантування. Стандарт *JPEG* навіть допускає використання власних таблиць квантування, які, однак, необхідно буде передавати декодеру разом зі стисненими даними, що збільшить загальний обсяг файлу. Зрозуміло, що користувачеві складно самостійно підібрати 64 коефіцієнти, тому стандарт *JPEG* використовує два підходи для матриць квантування. Перший полягає в тому, що у стандарт *JPEG* включені дві рекомендовані таблиці квантування: одна для яскравості, інша для кольору. Ці таблиці наведені нижче

Рекомендовані таблиці квантування

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	58	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Яскравість

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Колір

Другий підхід полягає у синтезі (обчисленні таблиці у процесі роботи) таблиці квантування, яка залежить від одного параметра *R*, заданого користувачем. На етапі квантування є змога керувати ступенем стискання і тут виникають найбільші втрати. Зрозуміло, що, задаючи таблиці квантування з великими коефіцієнтами, ми отримаємо більше нулів і, отже, більший ступінь стискання.

З квантуванням пов’язані й специфічні ефекти алгоритму. У разі великих значень кроку квантування втрати можуть бути настільки великими, що зображення розпадеться на однотонні квадрати розміром 8x8. Відповідно, втрати у високих частотах можуть виявитися у так званому ефекті Гібса, коли навколо контурів з різким переходом кольору утворюється хвилеподібний “німб”.

Крок 5. Переводимо матрицю 8x8 у 64-елементний вектор. На початку вектора, зазвичай, записуватимуться ненульові коефіцієнти, а в кінці утворюватимуться ланцюжки з нулів.

Крок 6. Перетворюємо вектор за допомогою модифікованого алгоритму RLE, на виході якого отримуємо пари типу (пропустити число), де “пропустити” є лічильником нулів, які пропускають, а “число” - значення, яке

необхідно поставити у наступну комірку.

Зазначимо, що перше число перетвореної компоненти *Y,* по суті, дорівнює середній яскравості блоку 8 х 8, його називають DC-коефіцієнтом. Аналогічно для всіх блоків зображення. Це наводить на думку, що коефіцієнти DC можна ефективно стиснути, якщо запам’ятати не їхні абсолютні значення, а відносні у вигляді різниці між DC-коефіцієнтом поточного блоку і DC-коефіцієнтом попереднього блоку, а перший коефіцієнт запам’ятати так, як він є. У цьому разі коефіцієнти DC можна впорядкувати. Решта коефіцієнтів, які називають AC-коефіцієнтами, зберігаються без змін.

Крок 7. Згортаємо отримані пари за допомогою нерівномірних кодів Хафмана з фіксованою таблицею. Причому для DC- і AC- коефіцієнтів використовують різні коди, тобто різні таблиці з кодами Хафмена.

Рисунок 6.1 - Структурна схема алгоритму JPEG.

Процес відтворення зображення в цьому алгоритмі повністю є симетричним. Метод дає змогу стискати зображення у 10-15 разів без помітних візуальних втрат.

Хоча алгоритм *JPEG* є стандартом ISO, формат його файлів не був зафіксований. Користуючись цим, виробники створюють свої, на жаль, несумісні між собою формати, і, отже, можуть змінити алгоритм. Наприклад, внутрішні таблиці алгоритму, рекомендовані ISO, замінюють на свої власні. Трапляються також варіанти *JPEG* для специфічних застосувань.

Суттєвими позитивними сторонами алгоритму є: 1) Задають ступінь стискання. 2) Вихідне кольорове зображення може мати 24 біти на точку.

Негативні сторони алгоритму:

1. Зі збільшенням ступеня стискання зображення розпадається на окремі квадрати 8 к вже зазначалося 8. Це пов’язано з тим, що відбуваються великі втрати в низьких частотах під час квантування, і відтворити вихідні дані стає неможливо.
2. Виникає *ефект Гібса* - ореоли по межах різких переходів кольорів.

Контрольні запитання

1. Що визначає обернена теорема посимвольного нерівномірного кодування?
2. Який критерій існування коду з середньою довжиною кодових слів, що дорівнює ентропії джерела? З чого він випливає?
3. У чому полягає алгоритм побудови оптимального коду Хафмена?
4. У чому полягає алгоритм побудови оптимального коду Шеннона-Фано?
5. Які переваги та недоліки використання оптимального кодування Шеннона-Фано і Хафмена?
6. У чому полягає алгоритм побудови коду Шеннона?
7. З чого випливає однозначна декодованість коду Шеннона?
8. У чому полягає алгоритм побудови коду Гільберта-Мура?
9. З чого випливає однозначна декодованість коду Гільберта-Мура?
10. Які коди називають блоковими? Що таке порядок блокового коду?
11. У чому полягає метод блокування повідомлень?
12. Як можна перейти від однієї моделі до іншої? Що таке формат моделі? Які формати застосовують на практиці?
13. Як відбувається стиснення зображення за допомогою алгоритму JPEG?
14. Завдяки чому відбувається стиснення зображення алгоритмом JPEG?
15. Що таке дискретне косинус-перетворення? З якою метою його застосовують?
16. Що таке квантування? Для чого його використовують?
17. Що таке “зиґзаґ” сканування? Для чого його проводять?
18. Які позитивні та негативні сторони алгоритм

Тема 7. ЗАВАДОЗАХИЩЕНЕ КОДУВАННЯ

Як вже зазначалося, фундаментальним поняттям в захисті від завад є інформаційна надлишковість. За К. Шеноном інформаційна надлишковість оцінюється як

$$D = 1 - \frac{H_{реал}}{H_{max}},$$

де $H_{реал} = -\sum_{i=1}^{2n} p_i \log_2 p_i$; n – розрядність слів при двійковому кодуванні.

H_{max} – максимальна ентропія, яка має місце, коли всі $p_i (i = 1,2,...,2^n)$ однакові, тобто $p_i = \frac{1}{2^n} = 2^{-n}$ і ентропія може бути визначена за формулою Хартлі

$$H_{max} = \log_2 2^n = n.$$

При будь якому *відхиленні* від рівномірного розподілу можна говорити про інформаційну надлишковість, і $D > 0$.

До речі, інформаційну надлишковість іноді зручніше вимірювати просто як $D' = H_{max} - H_{реал}$.

Нас буде цікавити випадок, коли деяка частина n-розрядних слів із загальної кількості $N = 2^n$ не використовується. Позначимо кількість таких слів через $N_з$ і будемо називати їх забороненими. Це означає, що деякі ймовірності $p_i = 0$ і $D > 0$. Підмножина слів, кількість яких N_g, є дозволеними словами і використовуються джерелом інформації для кодування повідомлень.

Очевидно, $N = N_g + N_з$. Очевидно також, що помилка в повідомленні може бути виявлена тоді і тільки тоді, коли вона перетворює дозволене слово в заборонене.

Для кожного дозволеного слова таких різних помилок може бути рівно $N_з$, а максимальна кількість всіх можливих помилок при заданому способі двійкового кодування

$$N_{E\,max} = 2^n - 1,$$

де 1 записана в рахунок випадку, коли помилок немає. Відносну кількість (“долю”) помилок, яку можна виявити при заданому способі кодування, обчислимо як

$$\delta = \frac{N_з}{2^n - 1}.$$

Обчислимо δ для таких випадків.

1. Код із однією перевіркою на парність.

$N = 2^n$; $N_g = 2^{n-1}$ (дозволені слова мають парну кількість одиниць – їх рівно половина).

$$\delta = \frac{2^n - 2^{n-1}}{2^n - 1} = \frac{2^{n-1}}{2^n - 1} \approx \frac{1}{2} \ (50\%)$$

2. Код, що має два додаткових розряда (код із контролем по модулю 3).

$$\delta = \frac{2^n - 2^{n-2}}{2^n - 1} \approx \frac{3}{4} \ (75\%).$$

3. Код із “повторенням“. Кожне слово передається джерелом двічі. Тоді

$$\delta = \frac{2^{2n} - 2^n}{2^{2n} - 1} = \frac{2^n(2^n - 1)}{2^{2n} - 1} \approx \frac{2^n \times 2^n}{2^n \times 2^n} = 1 \ (100\%)$$

З цих прикладів бачимо, що здатність коду до виявлення помилок зростає дуже швидко із зростанням надлишковості. Зауважимо також, що формула є універсальною в тому розумінні, що оцінка здатності до виявлення помилок не залежить від способу кодування інформації.

У класичній теорії кодування застосовується дещо інший підхід. Він базується на визначенні кодової відстані та понятті кратності помилки t. При двійковому кодуванні:

– кодова відстань d між двома словами – це кількість розрядів, якими відрізняється одне слово від другого;

– кратність помилки .**t** – це кількість розрядів, які спотворені помилкою;

– вектор помилки **E=(e_1,e_2,….,e_n)** .– це двійкова комбінація, яка містить 1 в тих розрядах слова, які спотворені.

Таким чином, кратність помилки – це “вага” вектора помилки – кількість одиниць в ньому.

Легко довести, що мінімальна кодова відстань для виявлення помилок кратності t і менше повинна бути

$$d_{min}^{вияв} = t_в + 1$$

При виконанні жодна помилка кратності **t** і менше не зможе перетворити одне дозволене слово в інше також дозволене слово.

Для ілюстрації розглянемо код із однією перевіркою на парність. Додавши один розряд таким чином, щоб кількість одиниць у слові стала парною, ми “розносимо” слова на $d_{min} = 2$ (до цього було $d_{min} = 1$). Зразу ж можна стверджувати, що будь-яка однократна помилка у цьому випадку буде виявлятися, бо вона перетворює слово із парною кількістю одиниць у слово із непарною кількістю. А які інші помилки будуть виявлятися? Очевидно, всі ті, для яких t непарне. А всього таких помилок буде приблизно половина (50%). В інших випадках потрібен детальний аналіз конкретних помилок, які виявляються і які не виявляються.

А тепер про більш складний випадок – виправлення помилок. Яка для цього потрібна надлишковість? І найголовніше, що потрібно для того, щоб помилку можна було не тільки виявити але й виправити? Очевидно, попередня умова, виконання якої необхідне для виявлення помилок, залишається справедливою і для їх виправлення: помилка повинна перетворювати дозволене слово в заборонене. Але цього не досить. Якщо два чи більше дозволених слова можуть переходити в одне заборонене, то, отримавши це слово замість дозволеного (без помилок), у отримувача не буде підстав, щоб надати перевагу будь-якій із гіпотез щодо того, яке саме дозволене слово було спотворено завадою. Що ж потрібно для того, щоб можна було таку первагу віддати? Очевидно, для надання такої переваги ймовірності помилок повинні бути суттєво різними. Тоді, отримавши заборонене слово, будемо мати підстави віддати перевагу тій помилці, яка має найбільшу ймовірність. Але де

взяти ці ймовірності? В реальних випадках вони не можуть вважатися відомими.

В теорії завадостійкого кодування вважається (і це може бути обґрунтовано теорією ймовірностей), що для незалежних помилок ймовірність виникнення помилки дуже швидко зменшується із зростанням її кратності, тобто найбільш ймовірні однократні помилки, менш ймовірні 2-кратні помилки і так далі.

У будь-якому випадку множина можливих помилок завжди має бути обмежена або конкретною кратністю, або конкретним переліком. Якщо таке обмеження виконано, (а це обов'язково треба зробити), то достатню умову для можливості виправлення помилок можна сформулювати так: *кожна з помилок, яка має бути виправлена, повинна переводити кожне із дозволених слів не більше, ніж в одне заборонене слово.*

Таким чином, кожному дозволеному слову треба мати можливість співставити не менше, ніж $N_E^{вип}$ заборонених слів, звідси безпосередньо витікає, що

$$N_g \cdot N_E^{вип} + N_g \le 2^n,$$

або $N_g \cdot N_E^{вип} \le 2^n - N_g = N_з$, звідки

$$N_E^{вип} \le \frac{N_з}{N_g}.$$

Для виправлення помилок певної кратності (частковий випадок) корегуючу здатність можна визначити через мінімальну кодову відстань

$$d_{min}^k = 2t_k + 1.$$

Групові коди. Із попереднього матеріалу може скластися враження, що знаючи наведены вище співвідношення, легко побудувати будь-який код. Наприклад, для цього, здавалось би, досить "рознести" дозволені слова на відповідну мінімальну відстань, і задачу можна вважати розв'язаною. Але це

лише на перший погляд. Дійсно, коли треба знайти, наприклад, тільки кілька сотень дозволених слів ($N_g \approx 2^5 ... 2^{10}$), то за допомогою комп'ютера це можна зробити, мабуть, за кілька хвилин. В результаті будемо мати таблицю дозволених слів, яку можна використати при декодуванні (виявленню або виправленню помилок). А що коли $N_g = 2^{20} ... 2^{100}$ або ще більше? Де зберігати таку таблицю, якщо її навіть вдасться побудувати? (На це потрібно витратити далеко не хвилини і навіть не години комп'ютерного часу, бо задача має комбінаторний характер).

Головна мета класичної теорії кодування полягає у створенні *конструктивних* процедур кодування і декодування, які мають прості апаратні або програмні реалізації. Теорія кодування вивчає головним чином так звані **групові коди**. Ця назва походить від поняття *група* з вищої алгебри.

Група – це замкнена відносно групової операції сукупність об'єктів. Група обов'язково містить також елемент, який називають одиничним або нульовим в залежності від того, яка використовується групова операція. Поняття замкненості полягає в тому, що застосування групової операції до двох чи більше елементів групи дає в результаті також елемент групи, а групова операція між будь-яким елементом групи і одиничним (нульовим) елементом дає в результаті сам елемент.

Все це – визначення на абстрактному рівні, і, мабуть, не зовсім зрозуміло, про що йде мова. Наведемо прості приклади, які зроблять ці визначення прозорими.

Всі дійсні числа – група відносно операції додавання $(+)$ із нульовим елементом **0**. Складаючи дійсні числа, ми завжди отримуємо теж дійсні числа (замкненість), а додавши **0** до будь-якого числа, ми отримуємо теж саме число. Відносно операції множення $(\times)$ з одиничним елементом **1** теж отримаємо замкнену групу. Інший приклад: числа, які без остачі діляться на **3**. Складаючи або перемножуючи такі числа, ми отримуємо числа, які без остачі діляться на **3**. Можна навести і інші численні приклади.

В теорії кодування двійкові n-розрядні слова $X_i = (x_1, x_2 ... x_n)$ – це група відносно операції порозрядного додавання по модулю **2** ($\oplus$), а нульовий елемент – кодове слово $X_D = (0,0,...,0)$.

Саме по собі таке визначення поки нічого не дає – проблема компактного представлення множини кодових слів зберігається. Але з математики відомо, що група може бути представлена своїм базисом. Базис – це мінімальна сукупність незалежних елементів, застосовуючи до яких групову операцію (до різних сполучень елементів базису), можна відтворити всю групу.

Наприклад,

Базис: $X_1 = 100; X_2 = 010; X_3 = 001.$

Похідгні: $X_1 \oplus X_2 = 110; X_1 \oplus X_3 = 101; X_2 \oplus X_3 = 011; X_1 \oplus X_2 \oplus X_3 = 111$

Повна сукупність:000,100, 010, 001, 110, 101, 011, 111.

Можна сказати, що базис “породжує” повну сукупність елементів групи плюс нульовий елемент, який повинен бути в групі за визначенням. Базис – *компакт групи.*

Наведений приклад ілюструє лише саму ідею представлення групи своїм базисом. Звичайно, щоб задати всі 3-розрядні комбінації, зовсім не потрібно вводити нові спеціальні поняття.

Візьмемо інший приклад:

100110	**100110**
010011	**010011**
001101	**001101**
	110101
Базис	**101011**
	011110
	111000
	000000
	Повна сукупність

(Звернемо увагу: ми задали 8 слів за допомогою трьох).

Базис зручно (у цьому переконаємося у подальшому) представляти відповідною матрицею. У нашому випадку

$$Q = \begin{Vmatrix} 100110 \\ 010011 \\ 001101 \end{Vmatrix}$$

Таку матрицю називають **породжуючою** (або твірною) і майже завжди позначають символом Q.

У більшості випадків процедура кодування полягає у доповненні вихідного (исходного – рос.) слова деякою кількістю додаткових (надлишкових або перевірочних) символів (розрядів). Будемо (як ми вже домовились) позначати кількість вихідних розрядів через k, а кількість додаткових – через $(n-k)$, тобто

Зауважимо, що значення перших k розрядів кодування повинні зберігатися. Це так звані **систематичні (роздільні**) коди.

А з цього безпосередньо витікає, що в породжуючій матриці Q перші k стовпців повинні утворювати одиничну квадратну матрицю рангу k. Тільки у цьому випадку перші k розрядів кодової комбінації зможуть приймати довільні значення. Виконання цієї вимоги є необхідним для того, щоб не накладати будь-яких обмежень на інформацію, що підлягає кодуванню. Таким чином,

$$Q = I_k A_{k,n-k},$$

де I_k - одинична квадратна матриця рангу k; $A_{k,n-k}$ - матриця розмірності $k \times (n-k)$.

Нагадаю, що Q задає множину кодових слів, а кожен рядок Q – це теж кодове слово. Тому всі вимоги до множини кодових слів повинні виконуватися і для рядків Q. Зокрема, якщо кодові слова повинні бути рознесені на відстань

d_{min} одне від одного, то ця ж сама вимога повинна виконуватись і для рядків Q. Для нашого прикладу можна переконатися, що як для Q, так і для повної множини кодових слів, породжених Q, $d_{min}=3$. (Згадаємо, що такий код спроможний корегувати всі однократні помилки). Тобто, виконавши певні вимоги до Q, ми забезпечуємо задані властивості для усієї множини кодових слів.

З точки зору математики процедура кодування еквівалентна матричному множенню довільного k - розрядного слова $X^{(k)}=(x_1,x_2,....x_k)$ на породжуючи матрицю Q. Результатом такого множення буде добуток у вигляді n-розрядного слова $X^{(n)}=(x_1,x_2,...x_k x_k,x_{k+1},...,x_n)$. Причому, $X^{(k)}$та $X^{(n)}$ треба розглядати як вектор-рядки, а операції множення і додавання виконуються за правилами алгебри логіки:

Проаналізуємо, що відбудеться при такому множенні:

$$\begin{Vmatrix} x_1 \\ x_2 \\ \dots \\ x_k \end{Vmatrix} \times \begin{Vmatrix} 10\dots0a_{11}a_{12}\dots a_{1,n-k} \\ 01\dots0a_{21}a_{22}\dots a_{2,n-k} \\ \\ 00\dots1a_{k1}a_{k2}\dots a_{k,n-k} \end{Vmatrix} = \begin{Vmatrix} x'_1 \\ x'_2 \\ \dots \\ x'_k \\ x_{k+1} \\ x_{k+2} \\ \dots \\ x_n \end{Vmatrix}$$

нагадаємо, що i-й розряд добутку – це скалярний добуток i- го розряду $X^{(k)}$ та i-го стовпця Q. Тобто

$$x'_1 = x_1$$
$$x'_2 = x_2$$
$$\dots$$
$$x'_k = x_k$$

$$\begin{cases} x'_{k+1} = a_{11}x_1 \oplus a_{21}x_2 \oplus \dots \oplus a_{k1}x_{k,} \\ x'_{k+2} = a_{12}x_1 \oplus a_{22}x_2 \oplus \dots \oplus a_{k2}x_{k,} \\ x'_n = a_{1,n-k}x_1 \oplus a_{2,n-k}x_2 \oplus \dots \oplus a_{k,n-k}x_{k,} \end{cases}$$

Останні $(n-k)$ рівнянь називаються рівняннями кодування. Вони, по суті, в явній формі визначають залежність значень додаткових розрядів від значень інформаційних.

В узагальненому вигляді кожне рівняння кодування можна записати як

$$x_{k+i} = a_{1i}x_1 \oplus a_{2i}x_2 \oplus ... \oplus a_{ki}x_k, \quad (i = 1,2,...n-k).$$

Це рівняння можна переписати так:

$$x_{k+i} = a_{1i}x_1 \oplus a_{2i}x_2 \oplus ... \oplus a_{ki}x_k = 0, \quad (i = 1,2,...n-k)$$

Треба підкреслити, що ми лише *переписали* рівняння, врахувавши особливості функції $\oplus$. Зокрема, ми маємо право переносити будь-який доданок з лівої частини в праву і навпаки без зміни знаку, бо від'ємні доданки не мають змісту.

До цього моменту ми ніяк не враховували помилки, і рівняння кодування виконується, коли помилок немає. А якщо вони виникнуть, то може вже і не виконуватися, тобто

$$x_{k+i} \oplus a_{1i}x_1 \oplus a_{2i}x_2 \oplus ... \oplus a_{ki}x_k = r_i, (i = 1,2,...n-k)$$

де r_i = **0**, якщо помилки немає або вона не виявляється, та $\mathbf{r_i=1}$, коли помилка є.

Ці рівняння називаються *рівняннями перевірки*, а r_i - результатом i-ї перевірки.

Сукупність $(n-k)$ результатів перевірки називається перевірочною послідовністю або **синдромом**

$$R = (r_1, r_2, ..., r_{n-k}).$$

З точки зору математики обчислення синдрому може бути записане як результат матричного множення n-розрядного слова на деяку матрицю H, яку називають перевірочною. Спосіб її утворення можна записати так:

$$Q = \left\| I_n A_{k,n-k} \right\|; \qquad H = A^T_{k,n-k} I_{n-k} \quad ,$$

де I_{n-k} – одинична квадратна матриця рангу $(n-k)$, $A^T_{k,n-k}$ – транспонована матриця $A_{k,n-k}$, тобто матриця, отримана із $A_{k,n-k}$ заміною рядків на стовпці:

$$A^T_{k,n-k} = \left\| \begin{matrix} a_{11}a_{12}...a_{1,n-k} \\ a_{21}a_{22}...a_{2,n-k} \\ a_{k1}a_{k2}...a_{k,n-k} \end{matrix} \right\|; A^T_{k,n-k} = \left\| \begin{matrix} a_{11}a_{21}...a_{k1} \\ a_{12}a_{22}...a_{k2} \\ a_{1,n-k}a_{2,n-k}...a_{k,n-k} \end{matrix} \right\|$$

Таким чином, перевірочна матриця H має $(n-k)$ рядків і n стовпців, а кожен рядок відповідає одній перевірці. З цього витікає, що результатом множення n-розрядного слова на H є саме синдром, тобто

$$R = X^{(n)} \times H$$

А тепер повернемось до власне помилок. А саме, яка відповідність між помилками та синдромами. Кодову комбінацію, яка спотворена помилкою, можна записати так:

$$X' = X \oplus E,$$

де X - комбінація без помилок, тобто та, яку передавало джерело інформації. Синдром, що відповідає X'

$$R = X' \times H = (X \oplus E) \times H = X \cdot H \oplus E \cdot H$$

Але за визначенням $X \cdot H = 0$ та

$$R = E \cdot H,$$

тобто синдром для певної конкретної помилки E_i можна отримати, помноживши вектор помилки E_i на перевірочну матрицю.

Ми вже знаємо, що в більшості випадків можна вважати, що найбільш ймовірними є однократні помилки. Тому, насамперед потрібно з'ясувати, який вигляд має відповідність між векторами однократних помилок і синдромами. Знайдемо синдроми, як відповідають однократним помилкам в n-розрядному кодовому слові.

$$R = E \times H = \begin{Vmatrix} e_1 \\ e_2 \\ \dots \\ e_i \\ \dots \\ e_n \end{Vmatrix} \times \begin{Vmatrix} \overbrace{a_{11}a_{21}\dots a_{k1}}^{k}\overbrace{10\dots 0}^{n-k} \\ a_{12}a_{22}\dots a_{k2}01\dots 0 \\ \\ a_{1,n-k}a_{2,n-k}\dots a_{k,n-k}00\dots 1 \end{Vmatrix}$$

Оскільки E – вектор однократної помилки, то серед його компонент є тільки одна одиниця. З цього витікає, що синдром помилки в i-му розряді кодового слова в точності співпадає з i-м стовпцем матриці H. Іншими словами, стовпці матриці H – це синдроми однократних помилок. Тому

$$E = (e_1, e_2, \dots, e_i, \dots, e_n) \Rightarrow R = (a_{i1}, a_{i2}, \dots, a_{in}),$$

де лише $e_i = 1$.

Тепер вже можна сформувати вимоги до перевірочної матриці групового коду для виявлення і виправлення помилок.

Виявлення помилок.

1. Помилка E_i виявляється кодом, якщо $E_i \times H \neq 0$.

2. Перевірочна матриця коду, що виявляє всі однократні помилки, повинна мати n ненульових стовпців.

У найпростішому випадку

$$H = \|11...1\|, \text{ і, відповідно,}$$

$$Q = \begin{Vmatrix} 10...01 \\ 01...01 \\ \\ 00...11 \end{Vmatrix}$$

– це код з однією перевіркою на парність, що має один додатковий символ

$$x_{k+1} = x_1 \oplus x_2 \oplus ... \oplus x_n .$$

Рівняння перевірки

$$R = r = x_{k+1} \oplus x_1 \oplus x_2 \oplus ... \oplus x_n .$$

Виправлення (корекція) помилок.

Очевидно, якщо код повинен виправляти сукупність помилок $E_1, E_2, ...E_i, ...E_e$, то всі відповідні цим помилкам синдроми повинні бути різними, тобто відповідність $E_i \leftrightarrow R_i$ має бути взаємно однозначною. Тільки в цьому випадку ми зможемо за значенням синдрому знайти відповідний йому вектор помилки і виправити помилку. З цього зокрема витікає, що перевірочна матриця коду, який виправляє всі однократні помилки, повинна мати **n** різних ненульових стовпців. Зокрема матриця, в якій стовпці пробігають всі ненульові значення $(n-k)$-розрядних чисел

$$H = \begin{Vmatrix} 000...11 \\ 000...11 \\ ... \\ 011...11 \\ 101...01 \end{Vmatrix}$$

може бути використана як перевірочна коду, який виправляє всі однократні помилки. Така матриця відрізняється від канонічної форми

$$H = \left\| A^T_{k,n-k} I_{n-k} \right\|$$

лише розташуванням стовпців. Якщо стовпці, які містять по одній одиниці, перенести праворуч і сформувати з них одиничну матрицю, то ми отримаємо саме канонічну форму. Слід зазначити, що така процедура еквівалентна лише перенумерації позицій символів і не є принциповою з точки зору властивостей коду.

Форма наведеної вище матриці є цікавою з іншого погляду. По-перше, саме у такому вигляді були запропоновані Хемінгом перші коди із корекцією помилок. По-друге, при використанні такої матриці синдром безпосередньо у двійковому вигляді вказує номер розряду, який спотворено:

E	$\Rightarrow$	R
100...00		00...01
010...00		00...10
001...00		00...11
...		...
000...01		11...11

У кодах Хемінга додаткові (перевірочні) розряди займають 1, 2, 8,...,2^{n-k} позиції. З точки зору матричного представлення це не дуже зручно, але з технічної – не викликає жодних додаткових проблем.

Хемінгом були запропоновані (1948р.) три класи кодів.

1. Уже розглянутий нами код з однією перевіркою на парність. Для нього $d_{min} = 2$ і він виявляє всі помилки непарної кратності. Для нього $H = \|111...1\|$.

2. Коди з виправленням однократних помилок з $d_{min} = 3$. Це ряд кодів, для яких

$$n - k = log_2(n+1)$$

Наприклад, код з $n = 7$ і $(n-k) = 3$

$$H = \begin{Vmatrix} 0001111 \\ 0110011 \\ 1010101 \end{Vmatrix}$$

Із перевірочної матриці можна викреслити деяку кількість стовпців і це не зменшить корегуючої здатності кода. Головне – не зменшувати кількість перевірочних розрядів. Такі коди (отримані шляхом викреслення деякої кількості стовпців із повної матриці) називають *укороченими.*

3. Ще один клас кодів Хемінга з $d_{min}=4$. Ці коди дозволяють виявляти помилки кратності $t=1,2,3$, або виправляти помилки кратності $t=1$ *та* виявляти помилки кратності $t=2$. Саме у цьому варіанті такі коди найчастіше і використовують.

Щоб отримати код з $d_{min}=4$, досить перевірочну матрицю H коду з $d_{min}=3$ доповнити ще одним рядком з одиниць:

$$H=\begin{Vmatrix} a_{11} & a_{12}...a_{1n} & 10...0 \\ a_{21} & a_{22}...a_{2n} & 01...0 \\ ... & ... & ... \\ a_{n-k,1} & a_{n-k,2}...a_{n-k,n} & 00...1 \\ 1 & 1...1 & 11...1 \end{Vmatrix}$$

Цьому додатковому рядку відповідає ще один $(n-k+1)$-ий додатковий розряд і ще одне рівняння перевірки

$$r_0 = x_1 \oplus x_2 \oplus ... \oplus x_{n+1}$$

Таким чином, при декодуванні ми отримуємо $(n-k+1)$ результатів перевірки

$$R=(r_1,r_2,...r_{n-k}) \text{ та } r_0.$$

Другий етап декодування коду базується на аналізі результатів перевірки. Вони можуть бути такими:

а) $R=(0,0,...0)$; $r_0=0$ – помилок немає;

б) $R\neq(0,0,...0)$; $r_0=0$ – однократна помилка, яка може бути виправлена на основі відповідності $R\to E$;

в) $R\neq(0,0,...0)$; $r_0=0$ $r_0=0$ – 2-кратна помилка, яка лише виявляється, але не може бути виправлена.

Таким чином, ми маємо універсальний метод побудови кодів з $d_{min}=2,3,4$, які, відповідно, виявляють однократні помилки, виправляють однократні та виявляють 2-кратні.

Для інших випадків не існує універсальних методів, тобто придатних для довільних значень n або k, побудови кодів. В узагальненому вигляді задача побудови коду формулюється так.

Задана сукупність векторів помилок $E_1,E_2,...,E_i,...,E_N$. Необхідно знайти таку перевірочну матрицю H, для якої всі добутки $E_i \times H = R_i \neq 0$ для випадку

виявлення заданої сукупності помилок та додатково у випадку їх виправлення всі синдроми повинні бути різними, тобто

$$R_i \neq R_j \qquad \text{для всіх } i, j = 1,2,...N .$$

В принципі не існує іншого алгоритму, крім прямого перебору, побудови кодів, що корегують (виявляють або виправляють) довільну сукупність помилок. При виконанні такого перебору може виявитися корисною така властивість: якщо вектору помилки E_i відповідає синдром R_i, а $E_j \to R_j$, то вектору помилки

$$(E_i \oplus E_j) \to (R_i \oplus R_j)$$

(ця властивість є наслідком лінійності всіх операцій, які застосовуються у теорії кодування).

Задача аналізу кодів, що виправляють помилки, досить складна тому, що потрібно аналізувати не окремі вектори помилки, а їх повну сукупність: всі синдроми, що відповідають такій сукупності, повинні бути різними. Як цього досягти для однократних помилок, ми вже розглянули. Для деяких інших сукупностей – розглянемо в наступному розділі в рамках циклічних кодів.

<u>Циклічні коди.</u> Насамперед зауважимо, що циклічні коди є підкласом групових кодів, тобто циклічні коди теж групові, але мають свою специфіку головним чином щодо математичного апарату та символіки, які використовуються для їх побудови та аналізу. В терміналах абстрактної алгебри циклічний код – це кільце, тобто така замкнена сукупність елементів, для яких задані дві операції та обернені до них. Зокрема, в нашому випадку це порозрядне додавання $\oplus$ (обернена операція співпадає із самою операцією) та множення (зворотна – ділення). Самі елементи – двійкові n-розрядні комбінації – записуються як поліноми. Правило, за яким встановлюється відповідність, можна пояснити прикладом:

$$\overbrace{\begin{array}{cccccccccc} 1 & 1 & 0 & 1 & ... & 1 & ... & 1 & 0 & 1 \\ n-1 & n-2 & n-3 & ... & ... & i & ... & 2 & 1 & 0 \end{array}}^{n \text{ розрядів}} \Rightarrow$$

Ст.полінома

$$\Rightarrow 1 \cdot x^{n+1} \oplus 1 \cdot x^{n-2} \oplus 0 \cdot 1 \cdot x^{n-3} \oplus ... \oplus x^i \oplus ... \oplus 1 \cdot x^2 \oplus 0x \oplus 1 \cdot x^0$$

Більш компактно цей поліном можна записати так:

$$F(x) = x^{n+1} \oplus x^{n-2} \oplus ... \oplus x^i \oplus ... \oplus x^2 \oplus 1$$

Такі поліноми можна додавати, множити і ділити за звичайними правилами з урахуванням специфіки операції суми по модулю 2, а саме

$$x^i \oplus x^i = 0;\ x^i \oplus x^i \oplus x^i = x^i$$

Наприклад,

а) $(x^5 + x^3 + x^2 + x) + (x^4 + x^3 + 1) = x^5 + x^4 + x^2 + x + 1$

б) $(x^5 + x^3 + x^2 + x)(x^4 + x^3 + 1) = x^9 + x^8 + x^7 + x^5 + x^4 + x^3 + x$

в)

$$\begin{array}{l} x^5 \oplus x^3 \oplus x^2 \oplus x \\ \underline{x^5 \oplus x^4 \qquad \oplus x} \\ x^4 \oplus x^3 \oplus x^2 \\ \underline{x^4 \oplus x^3 \oplus 1} \\ x^2 \oplus 1 - \textit{остача} \end{array} \qquad \begin{array}{l} x^4 + x^3 + 1 \\ \hline x \oplus 1 - \textit{частка} \end{array}$$

Можна дати таке (спрощене) визначення циклічного коду: циклічний код утворює множина дозволених кодових комбінацій, відповідні яким поліноми без остачі діляться на деякий поліном $P(x)$, який називають породжуючим. (Аналогія: множина чисел, які без остачі діляться на деяке число M Наприклад, всі числа кратні 2 (парні числа), або 3 чи іншому числу).

Існує два способи утворення циклічного коду, точніше, дозволених кодових комбінацій.

1. Множенням довільної k -розрядної комбінації у вигляді відповідного їй полінома $F(x)$ ступені $(k-1)$ на породжуючий поліном $P(x)$. У цьому випадку кодове слово може бути представлене як

$F(x) \cdot P(x)$

і, очевидно, воно завжди без остачі ділиться на $P(x)$.

Такий спосіб з математичної точки зору є найбільш простим і прозорим, але на практиці застосовується дуже рідко. Причина полягає в тому, що в цьому випадку ми отримуємо несистематичний (нерозділимий) код – інформаційна частина $F(x)$ у кодовій комбінації не зберігається (в результаті множення вона "змішується" з $P(x)$). Це суттєво ускладнює процедуру декодування при наявності помилок.

2. Дописуванням до $F(x)$ з боку молодших розрядів решти $R(x)$ від ділення $F(x) \cdot x^{n-k}$ на породжуючий поліном $P(x)$. (Зазначимо, що множення на x^{n-k} означає просте дописування до $F(x)$ $(n-k$ нулів.) Формально це можна записати так

$$F(x)\cdot x^{n-k} \oplus R(x).$$

Неважко переконати, що отриманий таким чином поліном завжди буде без решти ділитися на $P(x)$.

Як підкреслювалось на самому початку, циклічні коди є підкласом групових кодів, тобто будь який циклічний код може бути заданий також відповідно породжуючою матрицею Q. Для цього досить знайти решти для перших k рядків одиничної матриці I_n

$$Q = \begin{Vmatrix} 10...0R_1(x) \\ 01...0R_2(x) \\ \\ 00...1R_k(x) \end{Vmatrix}$$

Не будемо забувати, що рядки матриці Q – це насамперед кодові слова в вони можуть бути утворені за звичайними правилами. Утворимо їх для прикладу конкретного кода з $P(x) = x^3 \oplus x \oplus 1$ і $k = 4$.

Рядки одиничної матриці і відповідні операції:

$$1000 \to F_1(x) = x^3;\ F_1(x)\cdot x^3 = x^6 \to 101$$

$$0100 \to F_2(x) = x^2;\qquad F_2(x)\cdot x^3 = x^5 \to 111$$

$$0010 \to F_3(x) = x;\ F_3(x)\cdot x^3 = x^4 \to 110$$

$$0001 \to F_4(x) = 1;\ F_4(x)\cdot x^3 = x^3 \to 011$$

Таким чином, породжую чому поліному $P(x) = x^3 \oplus x \oplus 1$ для кода з $k = 4$ інформаційними розрядами відповідає породжуюча матриця

$$Q = \begin{Vmatrix} 1000101 \\ 0100111 \\ 0010110 \\ 0001011 \end{Vmatrix}.$$

Циклічний код з $P(x)$ і груповий код з Q еквівалентні. Крім того, як неважко переконатися, обчислення решти $R(x)$ еквівалентно обчисленню синдрому R в групових кодах. Зауважимо також, що відповідність між груповими і циклічними кодами не є взаємною: будь-якому циклічному коду можна знайти еквівалентний груповий код, але не навпаки. Клас групових кодів значно ширший, ніж клас циклічних кодів.

Основні циклічні коди. На практиці саме циклічні коди використовуються найчастіше. Це пояснюється їх простою схемною реалізацією на регістрах зсуву із зворотними зв'язками та простим математичним апаратом, за допомогою якого можна описати циклічні коди та проаналізувати їх корегуючу здатність.

Для побудови циклічних кодів найчастіше використовуються так звані неприводимі поліноми (аналог – прості числа). Із теорії кодування відомо, що далеко не всі неприводимі поліноми придатні для побудови ефективних кодів. Так, для отримання максимального числа різних решт при декодуванні породжуючий поліном $P(x)$ повинен бути дільником $x^n \oplus 1$. Таблиці неприводимих дільників для різних значень n наводяться у більшості книжок з теорії кодування. Не заглиблюючись в дебрі теорії, приймемо цю вимогу на віру і розглянемо деякі конкретні коди.

1. Код з $P(x) = x \oplus 1$ ми вже фактично розглянули. Цей код забезпечує $d_{min} = 2$ і дозволяє виявляти всі однократні помилки, а також всі помилки непарної кратності. Це легко довести, аналізуючи подільність полінома, що відповідає вектору помилки $E(x) = x^i \oplus x^j \oplus ... \oplus x^2$ (кількість доданків непарна), на $P(x)$.

2. Код з $P(x) = x^3 \oplus x \oplus 1$. Цей поліном неприводимий і є дільником $x^7 \oplus 1$, тому існує код з $n = 7$ та $(n-k) = 3$. Код з такими параметрами дає $d_{min} = 3$, тому він спроможний виправляти однократні помилки або виявляти двократні. Такі самі характеристики має код з $P = x^3 \oplus x^2 \oplus 1$. Із таблиць можемо знайти інші поліноми для інших значень n та k. Наприклад,

n	k	$P(x)$
15	11	$x^4 \oplus x \oplus 1$
31	26	$x^5 \oplus x \oplus 1$
63	57	$x^6 \oplus x \oplus 1$
127	120	$x^7 \oplus x^3 \oplus 1$
255	247	$x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$
...	...	...

Всі ці коди оптимальні (довершені) і є аналогами уже розглянутих нами кодів Хемінга. В таблиці наведено лише по одному поліному для кожного значення n та k (насправді може бути по кілька еквівалентних варіантів).

2. Код з $d_{min} = 4$ для виправлення однократних і виявлення двократних помилок. Такі коди отримують із попередніх домноженням $P(x)$ на $(x \oplus 1)$. При домноженні кількість перевірочних розрядів і, відповідно довжина кодових слів збільшується на 1. Наприклад, з (31,26) – коду отримаємо (32,26) – код із породжуючим поліномом.

$$P'(x) = P(x) \cdot (x \oplus 1)(x^5 \oplus x \oplus 1)(x \oplus 1) = x^6 \oplus x^2 \oplus \cancel{x} \oplus x^5 \oplus \cancel{x} \oplus 1 =$$
$$= x^6 \oplus x^5 \oplus x^2 \oplus 1$$

Цей код виправляє однократні та виявляє двократні помилки у 32 – розрядних двоїчних словах. Процедура декодування аналогічна відповідній процедурі для кодів Хемінга з $d_{min} = 4$. Ознакою двократної помилки у цьому випадку буде неподільність прийнятої комбінації на $P'(x)$ і подільність на $(x \oplus 1)$.

Коди БЧХ. Ці коди отримали свою назву від прізвищ їх авторів (Р. Боуз, Д. Рой-Чаудхурі та А. Хоквінгем). Коди БЧХ є узагальненням кодів Хемінга і дозволяють виправляти кратні помилки $(t = 2,3,4,...)$. Не заглиблюючись в теорію кодів БЧХ, зазначимо, що породжуючий поліном для цих кодів утворюється як найменше загальне кратне деякої сукупності непрводимих поліномів (у більшості випадків – це добуток поліномів). Деякі з кодів БЧХ наведені у таблиці.

$№_{n/n}$	n	k	t	$P(x)$
1	15	7	2	$x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1$
2	15	4	3	$x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1$
3	31	21	2	$x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5 \oplus x^3 \oplus 1$
4	31	16	3	$x^{15} \oplus x^{11} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus x \oplus 1$
5	63	51	2	$x^{12} \oplus x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^3 \oplus 1$
6	63	45	3	$x^{18} \oplus x^{17} \oplus x^{16} \oplus x^{15} \oplus x^9 \oplus x^7 \oplus x^6 \oplus x^3 \oplus x^2 \oplus x \oplus 1$
7	63	36	5	$x^{27} \oplus x^{22} \oplus x^{21} \oplus x^{19} \oplus x^{18} \oplus x^{17} \oplus x^{15} \oplus x^8 \oplus x^4 \oplus x \oplus 1$

Контрольні питання

1. Що таке корегуюча здатність завадозахищеного кода? Чим вона визначається?
2. Що спільного і чим відрізняються між собою групові та циклічні коди?
3. Яку має структуру породжуючи матриця коду?
4. Які способи утворення
5. Що таке неприводимі поліноми?
6. У чому полягає процедура декодування циклічних кодів?
7. У чому полягає основний спосіб утворення БЧХ – кодів?

Список використаної та рекомендованої літератури

1. Архипова О. О. Частотний аналіз використання букв української мови І О. О. Архипова, В. М. Журавльов 11 Радіоелектроніка. Інформатика. Управління. - 2009. - № 2(21). - С. 53-56.
2. Ван дер Варден Б. Л. Алгебра / Б. Л. ван дер Варден ; пер. с нем. А. А. Бельского; под ред. Ю. И. Мерзлякова. - М. : Наука, 1976. - 648 с.
3. Ватолин Д. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. - М.: ДИАЛОГ-МИФИ, 2003. - 384 с.
4. Галлагер Р. Теория информации и надежная связь / Р. Гал- лагер ; пер. с англ, под ред. М. С. Пинскера и Б. С. Цыбакова. - М.: Сов. радио, 1974. - 720 с.
5. Жураковсъкий Ю. П. Теорія інформації та кодування : підручник І Ю. П. Жураковський, В. П. Полторак. - К. : Вища школа, 2001. - 255 с.
6. Кларк Дж. мл. Кодирование с исправлением ошибок в системах цифровой связи І Дж. Кларк мл., Дж. Кейн ; пер. с англ, под ред. Б. С. Цыбакова. - М. : Радио и связь, 1987. - 392 с.
7. Котельников В. А. Теория потенциальной помехоустойчивости / В. А. Котельников. - М. : Госэнергоиздат, 1956. - 152 с.
8. Кузьмин И. В. Основы теории информации и кодирования / И. В. Кузьмин, В. А. Кедрус. - Киев : Вища школа, 1977. - 280 сеорія інформації
9. Ленг С. Алгебра / С. Ленг ; пер. с англ. Е. С. Голода; под ред. А.И. Кострикина. - М. : Мир, 1968. - 564 с.
10. Лидовский В. В. Теория информации : учеб, пособие /. Лидовский. - М. : Компания Спутник+, 2004. - 112 с.
11. Локазюк В.М., Савченко Ю.Г. Надійність, контроль, діагностика і модернізація ПК.-Навчальний посібник. Київ, Видавничий центр «Академія», 2004, 375 с.
12. Питерсон У. Коды, исправляющие ошибки / У. Питерсон, Э. Уэлдон ; пер. с

англ, под ред. Р. Л. Добрушина и

13. Подлевський Б.М.,Рикалюк Р.Є Теорія інформації, Львів, ЛНУ імені Івана Франка, 2016, 339 с.
14. Стратонович Р. Л. Теория информации / Р. Л. Стратонович. - М. : Сов. радио, 1975. - 424 с.
15. Сэломон Д. Сжатие данных, изображений и звука / Д. Сэломон ; пер. с англ. В. В. Чепыжова. - М.: Техносфера, 2004. - 368 с.
16. Туликова Н. О. Теорія інформації : навч. посібник / Н. О. Ту- лякова. - Суми : Вид-во СумДУ, 2008. - 212 с.
17. Фано Р. Передача информации. Статистическая теория связи / Р. Фано ; пер. с англ, под ред. Р. Л. Добрушина. - М. : Мир, 1965.-483 с.
18. Харкевич А. А. Очерки общей теории связи : Избр. тр. А. Харкевич. - М.: Наука, 1973. - Т. 3. - 194 с.
19. Хэмминг Р. В. Теория кодирования и теория информации / Р. В Хэмминг; пер. с англ. - М.: Радио и связь, 1983. - 176 с.
20. Цымбал В.П. Теория информации и кодирование / П. Цымбал. - Киев : Вища школа, 1992. - 263 с.
21. Шеннон К. Работы по теории информации и кибернетике / К. Шеннон; пер. с англ, под ред. О. Б. Лупанова и Р. Л. Добрушина. - М.: Иностр. л-ра, 1963. - 830 с.
22. Шульгин В. И. Основы теории передачи информации: учеб, пособие. 4.1. Экономное кодирование / В. И. Шульгин. - Харьков : Нац. аэрокосм, ун-т “Харьк. авиац. ин-т”, 2003. - 102 с.
23. Шульгин В. И. Основы теории передачи информации : учеб, пособие. Ч. 2. Помехоустойчивое кодирование / В. И. Шуль-
24. гин. - Харьков : Нац. аэрокосм, ун-т “Харьк. авиац. ин-т”, 2003.-87 с.
25. Яглом А. М. Вероятность и информация / А. М. Яглом, И. М. Яглом. — М.: Наука, 1973. — 511 с.
26. Элементы теории передачи дискретной информации / Л. П. Пуртов, А. С. Замрий, А. И. Захаров, В. М. Охорзин ; под ред. Л. П. Пуртова. - М. : Связь,

1972. - 232 с.

27. Storer J. A. Data Compression via Textual Substitution Z J. A. Storer, T. G. Szymanski // J. of ACM. — 1982. — Vol. 29, N 4. -P. 928-951.
28. Welch T. A. A Technique for High-Performance Data Compression I T. A. Welch // IEEE Computer. - 1984. - Vol. 17, N 6. - P. 8- 19.
29. Ziv J. An Universal Algorithm for Sequential Data Compression / J. Ziv, A. Lempel // IEEE Transactions on Information Theory. — 1977. - Vol. 23, N 3. - P. 337-343.
30. Ziv J. Compression of Individual Sequences via Variable Rate Coding / J. Ziv, A. Lempel // IEEE Transactions on Information Theory. - 1978. - Vol. 24, N 5. - P. 530-536.