МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА МОДЕЛЮВАННЯ

Алеся СІНЧУК

ТЕОРІЯ ІНФОРМАЦІЇ ТА КОДУВАННЯ

К У Р С Л Е К Ц І Й

для студентів спеціальностей:
1 1 3 - Прикладна математика, 122 - Комп'ютерні науки,
121 – Інженерія програмного забезпечення.

Рівне - 2023

С 53 **УДК 519.72[075.8]**
ББК 32.811я73

Рекомендовано до друку навчально-методичною комісією факультету математики та інформатики Рівненського державного гуманітарного університету (протокол № 3 від 19 квітня 2023 р.).

Рецензенти:
Сафоник А.П., д.т.н., професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки Національного університету водного господарства та природокористування;
Гладка О.М., к.т.н., доцент кафедри комп'ютерних технологій та економічної кібернетики навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки Національного університету водного водного господарства та природокористування.

Сінчук А. М.
С 53 Теорія інформації та кодування: [Курс лекцій| / А. М. Сінчук. - Рівне: РДГУ, 2023. - 82 с.

Курс лекцій присвячено основним методам теорії інформації та кодування, які широко використовуються в сучасних комп'ютерних інформаційних технологіях в різних сферах діяльності людини. В даному курсі вивчаються методи кодування інформації у системах зв'язку і обчислювальних системах, що дозволяють здійснити збереження, перетворення і передачу інформації з великою надійністю і малою надмірністю.

Орієнтована для студентів спеціальності 113 - Прикладна математика, 121 – Інженерія програмного забезпечення, 122 - Комп'ютерні науки.

С 53
УДК 519.72[075.8]
ББК 32.811я 73

ЗМІСТ

Вступ

Теорія інформації та кодування – це розділ прикладної дискретної математики і математичної кібернетики, в якому вивчаються методи кодування інформації у системах зв'язку і обчислювальних системах, що дозволяють здійснити збереження, перетворення і передачу інформації з великою надійністю і малою надмірністю.

Основна мета даної роботи є представлення основних методів теорії інформації і кодування, які широко використовуються в сучасних комп'ютерних інформаційних технологіях в різних сферах діяльності людей. В результаті вивчення курсу студенти повинні збагатити свої теоретичні знання та практичні навички у галузі застосування теорії інформації та кодування в комп'ютерних інформаційних технологіях; знати способи та мати практичні навички вимірювання кількості інформації, яку несуть в собі будь-які повідомлення, способи ефективного кодування таких повідомлень для передачі, збереження, обробки та відображення в інформаційних системах, забезпечення необхідної надійності їх роботи за рахунок використання завадостійкого кодування, вміти використовувати основні принципи кодування інформації з метою підвищення ефективності вводу, збереження, обробки та передачі інформації в сучасних інформаційних технологіях.

Курс лекцій з дисципліни «Теорія інформації та кодування» забезпечує вивчення, зокрема, таких дисциплін: «Контроль та діагностика інформаційних систем», «Системний аналіз та проектування систем обробки інформації», «Основи автоматизованого проектування складних об’єктів і систем», «Комп'ютерні мережі», «Проектування комп'ютерних систем і мереж» та ін.

ТЕМА 1. Елементи теорії інформації.

1.1. Поняття інформації

Дані – це сформульоване у вигляді наборів букв чи цифр відображення результатів людської діяльності чи розуміння нею навколишнього світу. Дані від джерела інформації передаються за допомогою *повідомлень*. *Інформацією* стають ті *повідомлення*, що знімають невизначеність, що існувала до їхнього надходження.

Приклад 1.1. Повідомлення ―Сьогодні ранком зійшло сонце‖ для нас не несе в собі нових даних і тому не є для нас інформацією. В той час повідомлення ―Вася кохає Олену‖ є інформацією для нас у випадку, якщо ми цього не знали до того.

Інформація — нематеріальна сутність, за допомогою якої з будь-якою точністю можна описувати реальні (матеріальні), віртуальні (можливі) і понятійні сутності. *Інформація* — протилежність невизначеності. Інформація може бути двох видів: дискретна (цифрова) і неперервна (аналогова). *Дискретна* інформація характеризується послідовністю значень деякої величини, а *неперервна* — неперервним процесом зміни деякої величини. Неперервну інформацію може, наприклад, давати датчик атмосферного тиску чи датчик швидкості автомашини. Дискретну інформацію можна одержати від будь-якого цифрового індикатора: електронного годинника, лічильника магнітофона і т.п.

Дискретна інформація зручніше для обробки людиною, але неперервна інформація частіше зустрічається на практиці, тому необхідно вміти перетворювати неперервну інформацію в дискретну і навпаки.

Дискретизація – процес переведення неперервної інформації в дискретну. При перекладі неперервної інформації в дискретну важлива так називана *частота дискретизації* ν, що визначає період ($T = 1/\nu$) визначення значення неперервної величини.

Чим вище частота дискретизації, тим точніше відбувається перетворення неперервної інформації в дискретну. Але з ростом цієї частоти росте і розмір дискретних даних, що отримуються при такому

перетворенні, і, отже, складність їхньої обробки, передачі і збереження. Для підвищення точності дискретизації необов'язково необмежено збільшувати її частоту, бо вона повинна бути не менш ніж у два рази вище найбільшої частоти гармоніки, що входить у величину, що дискретизується (будь-яка неперервна величина описується безліччю накладених один на одного хвильових процесів, які називаються *гармоніками*, що записуються у вигляді функцій виду: $A\sin(\omega t + f)$ *, де* A - це *амплітуда*, w - *частота*, t - *час* і f - *фаза гармоніки*).

Прикладом використання цієї теореми є лазерні компакт-диски, звукова інформація на яких зберігається у цифровій формі. Чим вище буде частота дискретизації, тим точніше будуть відтворюватися звуки і тем менше їх можна буде записати на один диск, але вухо звичайної людини здатне розрізняти звуки з частотою до 20 КГц, тому точно записувати звуки з більшою частотою безглуздо. Відповідно до теореми про вибірки частоту дискретизації потрібно вибрати не меншу за 40 КГц (у промисловому стандарті на компакт-диску використовується частота 44.1 КГц).

Приклад1. 2. У цифрових магнітофонах DAT частота дискретизації — 48 КГц. Яка максимальна частота звукових хвиль, які можна точно відтворювати на таких магнітофонах?

1.2. Кількість інформації в дискретному повідомленні. Ентропія

Припустимо, що джерело повідомлень може в кожен момент часу випадковим чином перейти в один із скінченої множини можливих станів. Таке джерело називають *дискретним джерелом повідомлень*.

Кожному стану джерела U ставиться у відповідність умовне позначення у вигляді знака.

Сукупність елементів $u_1, u_2, ..., u_i, ..., u_N$ які відповідають усім N можливим станам джерела називають його *алфавітом*, а кількість станів N *об'ємом алфавіту*.

Формування джерелом повідомлень зводиться до вибору їм деякого стану u_i і видачі відповідного елемента. Під *елементарним дискретним повідомленням* будемо розуміти символ u_i, що видається джерелом.

Протягом деякого часу T джерело може видати дискретне повідомлення у вигляді послідовності елементарних дискретних повідомлень, що представляють собою набір символів u_i (наприклад, u_5, u_1, u_3).

У загальному випадку окремі стани джерела можуть вибиратися їм частіше, інші рідше. Тому, в загальному випадку, джерело характеризується дискретним ансамблем U, тобто повною сукупністю станів з ймовірностями їхньої появи, що складають у сумі 1:

$$U = \begin{pmatrix} u_1 & u_2 & ..u_i & ...u_N \\ P(u_1) & P(u_2)...P(u_i)...P(u_N) \end{pmatrix}, \quad \sum_{i=1}^{N} P(u_i) = 1$$

де $P(u_i)$ – це ймовірність вибору джерелом стану u_i.

У кожному елементарному повідомленні міститься сукупність даних про стан дискретного джерела повідомлення. Визначаючи кількісну міру цієї інформації, ми не будемо враховувати її змісту і значення для конкретного одержувача. Очевидно, що при відсутності даних про стан джерела існує невизначеність щодо того, яке повідомлення u_i, з числа можливих, їм обрано, а при наявності цих даних невизначеність цілком зникає. Природно кількість інформації, що міститься в дискретному повідомленні вимірювати величиною зниклої невизначеності. Введемо міру цієї невизначеності, яку можна розглядати і як міру кількості інформації.

Дана міра повинна задовольняти ряд природних умов, одним з них є необхідність її монотонного зростання із збільшенням можливості вибору, тобто обсягу алфавіту джерела N. Крім того, необхідно, щоб міра, що вводиться, *мала властивість* адетивності, що заключається в наступному:

Якщо два незалежних джерела з обсягами алфавіту N і M розглядати як одне джерело, що одночасно реалізує пари станів n_i і m_i, то відповідно до принципу адитивності вважають, що невизначеність об'єднаного джерела дорівнює сумі невизначеностей вихідних джерел. Оскільки обсяги алфавіту об'єднаного джерела дорівнює $N \cdot M$, то шукана функція при рівній імовір-ності станів джерел повинна задовольняти умові $f(N \cdot M) = f(N) + f(M)$.

Перераховані вимоги виконуються, якщо за міру невизначеності джерела U з рівноймовірними станами, прийняти логарифм об'єму алфавіту джерела:

$$H(U) = \log N \qquad (1.2)$$

Легко бачити, що:

1) із збільшенням N величина $H(U)$ монотонно зростає.

2) у випадку якщо обсяг алфавіту джерела N дорівнює 1, тобто коли невизначеність відсутня, $H(U) = \log 1 = 0$.

3) величина $H(U)$ володіє властивістю адитивності, оскільки

$$\log(N \cdot M) = \log(N) + \log(M).$$

Вперше дана міра була запропонована Хартлі в 1928р. Основа логарифма в $H(U) = \log N$ не має принципового значення і визначає тільки одиницю кількості інформації. Найчастіше за основу використовують число 2, при цьому одиниця кількості інформації називається *бітом*, і являє собою інформацію, що міститься в одному дискретному повідомленні джерела рівноймовірних повідомлень з обсягом алфавіту рівним двом. При виборі в (1.2) основи логарифма рівною 10-ти одержуємо десяткову одиницю названу *дітом*. Іноді використовують натуральну одиницю кількості інформації – *нат* (основа логарифма дорівнює *e*).

Приклад 1.3. Щоб довідатися положення точки у системі з двох клітинок тобто одержати деяку інформацію, необхідно задати *1* питання ("Ліва чи права клітинка?"). Довідавшись положення точки, ми збільшуємо сумарну інформацію про систему на *1* біт ($I = \log_2 2$). Для системи з чотирьох клітинок необхідно задати 2 аналогічних

питання, а інформація дорівнює 2 бітам ($I = \log_2 4$). Якщо система має n різних станів, то максимальна кількість інформації дорівнює $I = \log_2 n$.

Іноді при розв'язуванні задач на знаходження кількості інформації зручно користуватися формулою Стірлінга (досить точна при $N > 100$): $N! \approx (N/e)^N$ та наслідком з неї: $\ln N! \approx N(\ln N - 1)$.

Розглянута міра кількості інформації може мати лише обмежене застосування, оскільки припускає рівну ймовірність вибору джерелом кожного з можливих його станів. У більш загальному випадку, ступінь невизначеності конкретного стану залежить не тільки від обсягу алфавіту джерела, але і від імовірності цього стану. У такій ситуації кількість інформації $i(u_k)$, що міститься в одному дискретному повідомленні u_k доцільно визначити як функцію імовірності появи цього повідомлення $P(u_k)$ і характеризувати величиною

$$i(u_k) = -\log P(u_k) = \log 1/P(u_k) . \qquad (1.3)$$

При цьому основа логарифма в (1.3) вибирається так же як і для (1.2). Знак мінус у першій рівності (1.3) необхідний для того, щоб $i(u_k)$ завжди була позитивним числом. Очевидно що, так само як і міра $H(U)$ величина $i(u_k)$ володіє властивістю адитивності і у випадку достовірного повідомлення, коли $P(u_k) = 1,\ i(u_k) = 0.$

У випадку, якщо джерело видає послідовність залежних між собою елементарних повідомлень і наявність попередніх повідомлень може змінити імовірність наступного а, отже, і кількість інформації в ньому, ймовірність появи повідомлення u_k має визначатися за умовною ймовірністю $P(u_k / u_{k-1}, u_{k-2}, \ldots).$ Тоді кількість інформації у u_k записується у вигляді:

$$i((u_k / u_{k-1}, u_{k-2}, \ldots) = -\log P(u_k / u_{k-1}, u_{k-2}, \ldots) \qquad (1.4)$$

Визначення (1.3) і (1.4) кількості інформації є випадковими величинами, оскільки самі повідомлення є випадковими.

Математичне сподівання кількості інформації в окремих повідомленнях *називається ентропією*:

$$H(U) = M\left\{\log\frac{1}{P(u_i)}\right\} = \sum_{i=1}^{N} P(u_i)\cdot\log\left(\frac{1}{P(u_i)}\right)$$

Вперше міра (1.5) була запропонована Клодом Шенноном у його фундаментальній роботі "Математичні основи теорії зв'язку" опублікованій в 1948-у році. Ця міра, була названа ентропією не випадково. Справа у тому, що вигляд формули (1.5) збігається з отриманим раніше фізиком Больцманом вираженням ентропії термодинамічної системи.

Розглянемо взаємозв'язок міри Шеннона з мірою Хартлі у випадку, якщо алфавіт джерела складають N рівноймовірних станів. Ймовірність кожного з них $P_i = 1/N$. В цьому випадку міри співпадають: $H(U) = \log(N) = -\log 1/N = -\log P_i$.

В загальному міру Хартлі можна трактувати, як *кількість інформації,* що припадає на одне дискретне повідомлення при умові їх рівноймовірності.

У той же час ентропія по Шеннону – це середня кількість інформації, яка міститься в одному з нерівноймовірносних станів. Вона дозволяє врахувати статистичні властивості джерела інформації.

1.3. Властивості ентропії.

1) Ентропія будь-якого дискретного ансамблю не від'ємна $H(U) \geq 0$. Рівність нулю можливо лише в тому випадку, коли джерело генерує одне єдине повідомлення з імовірністю $P = 1$. У цьому випадку ймовірності інших повідомлень дорівнюють нулю.

2) Нехай N - обсяг алфавіту дискретного джерела, тоді $H(U) \leq \log N$. Причому рівність має місце, коли всі повідомлення джерела рівноймовірні.

3) Ентропія об'єднання декількох незалежних статистичних джерел повідомлень дорівнює сумі ентропії вихідних джерел – властивість адитивності ентропії.

Приклад 1.4. Нехай маємо 192 монети, одна з яких фальшива. Визначимо скільки зважувань потрібно зробити, щоб визначити фальшиву монету. Якщо покласти на ваги рівну кількість монет, то одержимо 2 варіанти: *а*) ліва чашка нижче; *б*) права чашка нижче. Таким чином, кожне зважування дає кількість інформації $I = \log_2 2 = 1$ і, отже, для визначення фальшивої монети потрібно зробити не менш k зважувань, де k задовольняє умові $\log_2 2^k \geq \log_2 192$. Звідси, $K \geq 8$ або $k = 8$. Отже, нам необхідно зробити не менше *8-ми* зважувань (досить восьми).

Приклад 1.5. З'ясуємо, скільки біт інформації несе кожне двозначне число з усіма значущими цифрами (відволікаючи при цьому від його конкретного числового значення). Таких чисел може бути всього *90 (10 - 99),* тобто кількість інформації буде дорівнювати $I = \log_2 90$ або приблизно $I = 6.5$. В таких числах значуща перша цифра може приймати *9* значень *(1- 9),* а друга - *10* значень *(0-9),* тобто $I = \log_2 90 = \log_2 9 + \log_2 10$.

Приклад 1.6. Нехай розглядається алфавіт із двох символів російської мови - *«б»* і *«а»*. Відносні частоти зустрічання цих букв у частотному словнику російської мови рівні відповідно $p_1 = 0.028$, $p_2 = 0.062$. Візьмемо довільне слово p довжини N з k букв *«б»* і m ($k + m = N$) букв *«а»* над цим алфавітом. Число всіх таких можливих слів, як це випливає з комбінаторики, дорівнює $n = N!/(k!m!)$. Оцінимо кількість інформації в такому слові: $I = \log_2 n = \ln(n) / \ln 2 = \log_2 e\left[\ln N! - \ln k! - \ln m!\right]$. Скориставшись наслідком приведеною вище формулою Стирлінга отримуємо оцінку кількості інформації (у бітах) на *1* символ будь-якого слова:

$$I_I = I/N \approx \approx (\log_2 e / N) \times \left[(k+m)(\ln N - 1) - k(\ln k - 1) - m(\ln m - 1)\right] =$$

$$(\log_2 e / N) \times$$

$$\times\left[k \ln(N/k) - m\ln(N/m)\right] = -\log_2 e\left[(k/N)\ln(k/N) + (m/N)\ln(m/N)\right] \leq$$

$$\leq -\log_2 e\left[p_1 \ln p_1 + p_2 \ln p_2\right] = -\log_2 e\left[0.028\ln 0.028 + 0.062\ln 0.062\right] \approx 0.235 .$$

Приклад 1.7. У повідомленні *4* букви *“а”*, *2* букви *–б”*, *1* буква *–і”*, *6* букв *–р‖*. Визначимо кількість інформації в одному такому (із усіх можливих) повідомлень. Число N різних можливих повідомлень довжиною в *13* букв буде дорівнює величині: $N = 13!/(4!\cdot 2!\cdot 1!\cdot 6!) = 180180$. Кількість інформації в одному повідомленні буде дорівнює величині: $I = \log_2(N) = \log_2 180180 \approx 17.5$ *(біт).*

ТЕМА 2. Надлишкові коди і принцип використання надлишку

2.1. Модель системи передачі інформації. Канал зв'язку.

Повідомлення передаються від об'єкта до адресата за допомогою сукупності технічних засобів, що утворюють систему передачі інформації. Скільки існує методів відображення інформації, стільки можна створити і способів її передачі, а отже, і систем передачі інформації.

Розглянемо *модель типової цифрової системи зв'язку* з використанням кодів, що виправляють помилки - окремий випадок узагальненої моделі передачі інформації (рис. 2.1).

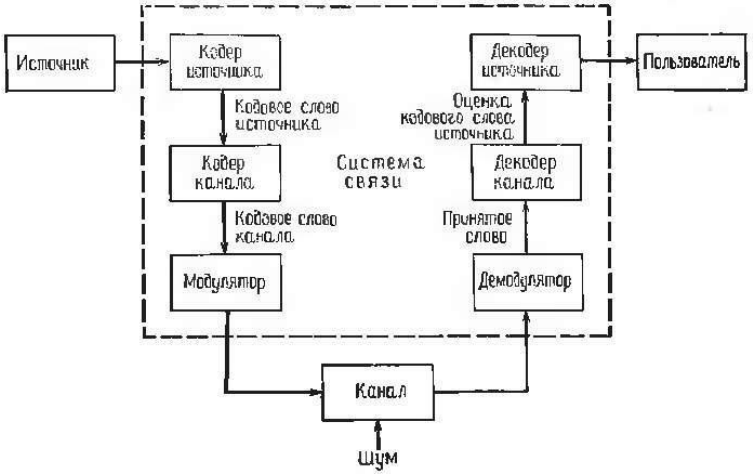


Рис. 2.1. Блок-схема цифрової системи зв’язку

Цифрова система зв'язку з'єднує джерело даних з одержувачем даних за допомогою каналу. Прикладами каналів є мікрохвильові лінії, коаксіальні кабелі, телефонні мережі.

Дані, що надходять у систему від джерела даних, насамперед обробляються кодером джерела, призначеним для кодування вхідної інформації в двійкові символи. Це проміжне представлення даних джерела називається *кодовим словом джерела*.

Далі дані обробляються кодером каналу, що перетворює послідовність двійкових символів кодового слова джерела в іншу послідовність двійкових символів, яку називають *кодовим словом каналу*. Кодове слово каналу являє собою нову, більш довгу послідовність з більшою, ніж у кодового слова надмірністю. Потім модулятор перетворить кожен символ кодового слова каналу у відповідний аналоговий символ з скінченої множини допустимих аналогових символів (двійкові символи перетворюються в сигнали). Послідовність сигналів передається по каналу. Через те, що в каналі виникають різного типу шуми, перекручування і інтерференція, вихідні сигнали каналу відрізняються від його вхідних сигналів. Демодулятор перетворює послідовність отриманих сигналів у послідовність символів одного з кодових слів каналу. Через шум у каналі демодулятор робить помилки. Демодульована послідовність символів називається *прийнятим словом*. Через помилки символи прийнятого слова не завжди відповідають символам кодового слова каналу.

Декодер каналу використовує надмірність кодового слова каналу для того, щоб виправити помилки в прийнятому слові, і потім видає оцінку кодового слова джерела. Якщо всі помилки виправлені, то оцінка кодового слова джерела збігається з вихідним кодовим словом джерела. Декодер джерела виконує операцію, зворотну операції кодера джерела, результат якої надходить до одержувача.

2.2. Перешкодостійкі коди. Принцип побудови перешкодостійких кодів

Проблема підвищення вірності передачі даних обумовлена невідповідністю між вимогами до якості передачі даних і якістю реальних каналів зв'язку. У мережах передачі даних необхідно забезпечити вірність не гірше 10^{-6} - 10^{-9}, а при використанні реальних каналів зв'язку і простого (первинного) коду зазначена вірність не перевищує 10^{-2} - 10^{-5}.

Одним з шляхів рішення задачі підвищення вірності в наш час є використання спеціальних процедур, заснованих на застосуванні перешкодостійких (коригуючих) кодів.

Прості коди характеризуються тим, що для передачі інформації використовуються всі кодові слова (комбінації), кількість яких дорівнює $N = q^n$ (q - основа коду, а n - довжина коду). У загальному випадку вони можуть відрізнятися одне від одного одним символом (елементом). Тому навіть один помилково прийнятий символ приводить до заміни одного кодового слова іншим і, отже, до неправильного прийому повідомлення в цілому.

Перешкодостійкими називаються коди, що дозволяють виявляти і(або) виправляти помилки в кодових словах, що виникають при передачі по каналах зв'язку. Ці коди будуються таким чином, що для передачі повідомлення використовується лише частина кодових слів, які відрізняються одне від одного більш ніж в одному символі. Ці кодові слова називаються *дозволеними*. Всі інші кодові слова не використовуються і відносяться до числа *заборонених*.

Задача кодування полягає в одержанні при передачі для кожної k- елементної комбінації з множини q^k відповідного їй кодового слова довжиною n з множини q^n .

Завдання декодування полягає в одержанні k -елементної комбінації з прийнятого n -розрядного кодового слова при одночасному виявленні чи виправленні помилок.

Надалі будемо розглядати тільки двійкові коди $(q = 2)$.

Приклад 2.1. Для передачі інформації використовуються наступні кодові комбінації, відмінні не менш, ніж двома розрядами: 0011, 0110, 1001, 1010, 1111, 0101, 0000. Нехай при передачі комбінації (наприклад, 0011) вийшла одинична помилка, в результаті чого спотворився перший розряд і прийнята комбінація 1011. Ця комбінація є забороненою. Це свідчить про наявність у ній помилки.

Приклад 2.2. Нехай дозволені комбінації відрізняються чотирма розрядами: 1100 і 0011. Легко переконатися, що в цьому випадку виявляються одно-, дво- і трьохкратні помилки, не виявляються лише чотирьохкратні помилки.

Основні параметри перешкодостійких кодів:

довжина коду - n ; довжина інформаційної послідовності - k ;

довжина перевірочної послідовності - $r = n - k$; кодова відстань коду - d_0 ; швидкість коду - $R = k/n$.

Зв'язок коректуючої здатності коду з кодовою відстанню: кодова відстань між двома кодовими словами (*відстань Хеммінга*) - це число позицій, у яких вони відрізняються одне від одного.

Приклад 2.3. Побудувати код з n=3, призначений для виявлення а) всіх однократних б) всіх двократних помилок.

а) Згідно (1) вибираємо $d_{min} = 2$. Кодові комбінації даного коду:

$(V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8) =$

(000,001,010,011,100,101,110,111).

Кодова матриця відстаней виглядає наступним чином:

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

З матриці відстаней, у якості дозволених комбінацій вибираємо наступні: $V_1 = 000$; $V_4 = 011$; $V_6 = 101$; $V_7 = 110$; або $V_2 = 001$; $V_3 = 010$; $V_5 = 100$; $V_8 = 111$.

б) Для виявлення двохкратних помилок кодова відстань $d_{\min} = 3$. У цьому випадку в якості дозволених комбінацій можна вибрати $V_1 = 000$ чи $V_8 = 111$, або $V_2 = 001$ і $V_7 = 110$, або $V_3 = 010$ і $V_6 = 101$ або $V_4 = 011$ і $V_5 = 100$.

Приклад 2.4. Знайти відстань Хеммінга між двома кодовими словами 10011100 та 01001111. Просумуємо попарно розряди кодових слів за модулем 2: 10011100 ⊗ 01001111 = 11010011. Результуюча сума у своєму складі містить 5 одиниць. Отже, відстань між словами за Хеммінгом дорівнює 5-ти.

Для оцінки ступеня відмінності між довільними комбінаціями коду використовується поняття *мінімальної кодової відстані* $d_{\min}$.

Для виявлення всіх помилок кратності t_p

задовольняти наступну умову: $d_{min} \geq t_0 + 1$.

Для виправлення помилок кратності t кодова відстань повинна задовільняти умову:

$$d_{min} \geq 2t + 1 \qquad (2.2)$$

Мінімальна кодова відстань $d_{мін} \geq 2*1+1=3$. Вибираємо в якості першої дозволеної комбінації $V_2 = 001$.При наявності однократних помилок дана комбінація може перейти в одну з заборонених комбінацій $V_6 = 101$, $V_4 = 011$, $V_1 = 000$, які можна прийняти в якості підмножини заборонених комбінацій вектора V_2, тобто у випадку прийняття одної з комбінації цієї підмножини, виноситься висновок, що переданий вектор V_2. Нехай у якості другої дозволеної комбінації вибирається вектор, який відноситься від першого на відстані $d = 2$, $V_5 = 100$. Йому повинна відповідати підмножина заборонених комбінацій $V_1 = 000$, $V_7 = 110$. Одержали перетин підмножин. При прийнятті заборонених символів V_1 чи V_6 не можна однозначно встановити, який був переданий сигнал – V_2 чи V_5. Коли ж в якості другої дозволеної комбінації вибрати комбінацію, яка відрізняється від V_2 на $d = 3$, тобто комбінацію $V_7 = 110$, якій відповідає підмножина заборонених комбінацій $V_3 = 010$, $V_5 = 100$, $V_8 = 111$, то підмножини заборонених комбінацій не перетинаються. Звідси, $d_{min} = 3$, забезпечує виправлення всіх одноразових помилок.

Для виправлення всіх помилок кратності до t_i і одночасного виявлення всіх помилок кратності не більше t_0 (при $t_0 \geq t_i$) кодова відстань винна задовольняти умову:

$$d_{min} \geq t_0 + t_i + 1 \qquad (2.3)$$

Ідея побудови коду з даною коректуючою властивістю, з вище зазначеного, заключається у внесенні в нього такого надлишку, який забезпечив би відстані між любими кодовими комбінаціями даного коду не менше d_{min}. На жаль, питання про визначення мінімальної

кількості надлишкових символів на сьогодні не вирішено. Існує лише ряд верхніх і нижніх оцінок (границь):

1. Границя Хеммінга $r \geq log_2 \left(1 + \sum_{i=1}^{d-1/2} C_n^i \right)$; (2.4)

2. Границя Плоткіна $r \geq 2d - 2 - log_2\, d$; (2.5)

3. Границя Варшамова-Гільберта $r \geq log_2 \left(1 + \sum_{i=1}^{d-1} C_n^{\,i} \right)$. (2.6)

Експериментально встановлено, що найбільш близькі значення дає границя Варшамова-Гільберта.

Всі наведені границі дозволяють зробити висновок, що при виконанні однієї із умов (4-6) є можливість побудувати код з даними параметрами.

Для кодів з $d_{min} = 3$ знайдене точне співвідношення між числом перевірочних символів r і довжиною коду n: $r \geq log_2(n+1)$

Класифікація кодів, які виправляють помилки

Коди, які виправляють помилки поділяються на блочні і неперервні.

В кодері, розробленому для вироблення блочних кодів, безперервна послідовність інформаційних символів розбивається на відрізки, що містять по k символів (блоки). Надалі операції виконуються над кожним блоком окремо незалежно від інших відповідно до обраного коду.

При використанні неперервних кодів, інформаційна послідовність піддається обробці без попередньої її розбивки на незалежні блоки. У кодері цього типу, інформація обробляється безперервно і кожній довгій (можливо, напівнескінченній) інформаційній послідовності ставиться у відповідність кодова послідовність, що складається з трохи більшої кількості символів. В кодері послідовність, що надходить на його вхід, розбивається на кадри з k_0 символів, де k_0 – зазвичай невелике число. Пізніше на

основі цих k_0 символів і попередніх кадрів утворюється блок довжиною n_0 з символів кодової послідовності.

Блочні коди поділяються на систематичні і несистематичні.

Систематичні блочні коди відрізняються від несистематичних тим, що у кожному кодовому блоці з n символів спочатку розташовуються k інформаційних символів, а потім $n - k$ перевірочних.

Більшість блочних і неперервних кодів складають *лінійні коди*. У цих кодів перевірочні символи визначаються в результаті проведення лінійних операцій над визначеними інформаційними символами (для випадку двійкових кодів найчастіше використовується сума по модулі два).

ТЕМА 3. Найпростіші коди та їх характеристики

Код з однією перевіркою на парність.

Даний код незалежно від кодової комбінації містить всього один перевірочний символ. Цей символ вибирається таким, щоб його сума по модулю два з усіма інформаційними символами дорівнювала нулю.

Через такий спосіб вибору перевірочного символа, кодова комбінація містить парне число одиниць. Наприклад, прості комбінації 00101 і 10101 при кодуванні їх кодом з однією перевіркою на парність виглядають відповідно 001010 і 101011. Ознакою спотворення кодової комбінації є непарність одиниць в прийнятій комбінації.

Даний код має $d_{min} = 2$, а тому дозволяє виявляти всі однократні помилки. Крім того даний код дозволяє виявляти всі помилки непарної кратності (трикратні, п'ятикратні, ...), так як тільки в цих випадках кількість одиниць в комбінації стане непарною.

Коефіцієнт надлишку для даного коду $R = r / n = 1 / n$.

Розподіл робочих кодових векторів по кодовим відстаням для всіх векторів однаковий і визначається за формулою:

$$N_{Д}(d_i) = C_n^{d_i}, \qquad (1)$$

де d_i приймає значення 2,4,6…n , коли n парне і 2,4,6,…, n -1 , коли n – непарне.

Коефіцієнт невірних переходів $K_{Н}(d)$ дорівнює:

$$K_{Н}(d) = N_{Д}(d_i) / N(d) = C_n^{d_i} / C_n^{d}, \qquad (2)$$

де $d = 1,2,3,...,n$ ($N(d)$ – кількість кодових слів довжиною n з кодовою відстанню d від заданого).

Ймовірність неправильного прийому кодової комбінації визначається ймовірністю появи невиявлених (парних) помилок.

Нехай в прийнятій комбінації спотворюються два певних символи, а інші не спотворюються. Ймовірність такої події дорівнює $P_e^2(1-P_e)^{n-2}$, де P_e – ймовірність збою одиничного символа при його передачі. Так як таких варіантів буде C_n^2, ймовірність двохкратних помилок $P_2 = C_n^2 P_e^2 (1-P_e)^{n-2}$.

Аналогічно можна визначити ймовірність появи чотирьохкратних помилок $P_4 = C_n^4 P_e^4 (1-P_e)^{n-4}$ і т.д.

Сумарна ймовірність появи невиявлених (парних) помилок:

$$P_{Н.П} = P_2 + P_4 + ... = сума_{i=2^n} C_n^i P_e^i (1-P_e)^{n-i},$$

де $n = 2,4,6, ..., n'$ (n' – парне найближче до n число).

Так як з підвищенням кратності ймовірність помилок різко падає і можна оцінювати ймовірність появи помилки за формулою:

$$P_{Н.П} = C_n^2 P_e^2 (1-P_e)^{n-2} \qquad (3)$$

Приклад. Знайти характеристики двійкового коду з однією перевіркою на парність для $n = 6$. Ймовірність збою одиничного символа $P_e = 10^{-4}$.

Число робочих комбінацій коду $N_p = 2^k = 2^{n-1} = 2^5 = 32$. Надлишок коду $R = 1/6 = 0,167$.

Розподіл робочих комбінацій по кодовим відстаням $N_Д(d)$ і коефіцієнт невірних переходів $K_H(d)$ знайдемо за формулами (1)-(2):

d	1	2	3	4	5	6
d_i	-	2	-	4	-	6
$N_Д(d_i)$	-	15	-	15	-	1
N(d)	6	15	20	15	6	1
$K_H(d)$	0	1	0	1	0	1

Значення $K_H(d)$ показують, що виявляються всі помилки непарної кратності ($K_H(d)=0$), а всі помилки парної кратності призводять до невірних переходів.

Ймовірність появи невиявленої помилки $P_{Н.П} = C_6^2 10^{-8}\left(1-10^{-8}\right)^4 = 1.5\times10^{-7}$.

Код з простим повторенням

В основу побудови коду з простим повторенням покладений метод повторення початкової кодової комбінації. Декодування проводиться шляхом порівняння першої (інформаційної) і другої (перевірочної) частин коду. При відсутності такого співпадання комбінація бракується (тобто виявляється помилка). В цьому випадку $k = n/2$, $r = n/2$. Потужність коду $N_P = 2^{n/2}$. Надлишок $R = r/n = n/2/n = 0{,}5$. Мінімальна кодова відстань $d_{min} = 2$.

Розподіл робочих комбінацій по кодовим відстаням дорівнює

$$N_P(d_i) = C_{n/2}^{d_i/2} \qquad (4)$$

де $d_i = 2,4,6,..., n/2$.

Коефіцієнт невірних переходів дорівнює

$$K_H(d) = C_{n/2}^{d/2} / C_n^d \qquad (5).$$

Ймовірність появи невиявлених помилок така ж, як і у коду з перевіркою на парність, але захищеність від перешкод цього коду

вища, так як він дозволяє виявляти всі помилки, за виключенням помилок в „парних‖ елементах (елементах, які стоять на одних і тих же позиціях в першій і другій комбінаціях). Даний код ефективний при дії пакетів помилок.

Приклад. Визначити характеристики коду з повторенням для $n = 6$. Розподіл робочих комбінацій по кодовим відстаням та коефіцієнти невірних переходів визначаємо за формулами (4) і (5):

d	1	2	3	4	5	6
d_i	-	2	-	4	-	6
$N_P(d_i)$	-	3	-	3	-	1
N(d)	6	15	20	15	6	1
$K_H(d)$	0	0.2	0	0.2	0	1

Звідки видно, що даний код дозволяє виявити всі помилки непарної кратності, 80% двохкратних помилок і 80% чотирьохкратних помилок. Лише всі шестикратні помилки приводять до невірних переходів комбінацій, які залишаються невиявленими.

Кореляційний код

Різновидом коду з повторенням є кореляційний код чи код з подвоєнням елементів, який характеризується введенням додаткових символів для кожного розряду інформаційної частини. Коли в розряді інформаційної частини стоїть 0, то в кореляційному коді цей розряд записується символами 01, коли 1 – символами 10. Наприклад, комбінація 10101 буде представлена у вигляді 1001100110. Показником спотворення коду є поява в „парних‖ елементах поєднань виду 00 чи 11.

Характеристики кореляційного коду повністю співпадають з характеристиками коду з простим повторенням. Різниця в стійкості проти перешкод кодових сигналів виявляється лише для несиметричних каналів, у яких ймовірності переходів $0 \Rightarrow 1$ і $1 \Rightarrow 0$

різні, а також для каналів, які мають різні ймовірності спотворення сусідніх символів в порівнянні з іншими можливими спотвореннями.

Інверсний код

Особливим різновидом коду з повторенням є інверсний код (код Бауера). Відмінність цього коду від звичайного коду з простим повторенням заключається в тому, що в тих випадках, коли вихідна комбінація містить парне число одиниць, друга комбінація точно повторює вихідну. Коли ж початкова комбінація містить непарне число одиниць, повторення проходить в інвертованому вигляді. Наприклад, комбінації 01100 і 10110 інверсним кодом представляються як 0110001100 і 1011001001.

Кодову комбінацію перевіряють у наступній послідовності. Спочатку сумують одиниці першої комбінації. Якщо їх число є парним, елементи додаткової комбінації приймають в незмінному вигляді. Далі обидві комбінації порівнюють поелементно, при виявленні хоча б одного неспівпадання, комбінація бракується. Коли ж кількість одиниць першої комбінації непарна, елементи другої комбінації приймають в інвертованому виді. Потім, як і в попередньому випадку, першу і другу комбінації порівнюють поелементно.

Така побудова коду дозволяє виявити практично всі помилки, за виключенням одночасного спотворення двох, чотирьох і т.д. елементів в початковій комбінації і відповідних їм двох, чотирьох і т.д. елементів в комбінації-повторенні.

Потужність коду $N_P = 2^{n/2}$. Коефіцієнт надлишку R не залежить від числа елементів і дорівнює 0,5. Даний код має

$$2, n/2 = 2;$$

мінімальну кодову відстань $d_{min} = 3, n/2 = 3;$

$$4, n/2 = 4.$$

Найбільш ймовірним видом невиявлених помилок є одночасне спотворення двох символів в початковій комбінації і відповідних їм двох символів в повторюваній комбінації. Ймовірність одночасного

спотворення якої-небудь пари символів в початковій комбінації дорівнює $P_2 = C^2_{n/2} P^2_e (1-P_e)^{n/2-2}$.

Ймовірність одночасного спотворення двох пар відповідних символів, тобто ймовірність появи невиявленої помилки, дорівнює

$$P_{H.П} = \left[C^2_{n/2} P^2_e (1-P_e)^{n/2-2}\right]^2 = \sim \left(C^2_{n/2}\right)^2 P^4_e \quad \textbf{(6).}$$

Виразу для розподілу робочих кодів по кодовим відстаням в загальному вигляді на даний час не для даного коду не одержано. Для знаходження розподілу кодів по кодовим відстаням слід визначити його для якого-небудь одного кодового вектора і поширити на всі інші.

Приклад. Побудувати інверсний код для передачі семи повідомлень і визначити його характеристики.

Кількість інформаційних символів $k = log_2 N_p \approx 3$. Розглянемо всі можливі комбінації нулів та одиниць довжиною в три символи за виключенням нульової комбінації (їх буде рівно 7). Для кожної з таких комбінацій утворимо кодове слово інверсного коду з довжиною $n = 2*k = 2*3 = 6$. Отримаємо 7 можливих кодових слів:

$V_1 = 100011;$ $V_2 = 010101;$ $V_3 = 001110;$ $V_4 = 110110;$

$V_5 = 101101;$ $V_6 = 011011;$ $V_7 = 111000.$

Розподіл кодових відстаней для кодового вектора, наприклад V_1, має вигляд: $V_1 V_2 V_3 V_4 V_5 V_6 V_7$

4 4 3 3 3 4.

Звідси $Np(1) = Np(2) = Np(5) = Np(6) = 0;$ $Np(3) = 3;$ $Np(4) = 3.$

Коефіцієнти невірних переходів $\hat{E}K_H(1) = K_H(2) = K_H(5) =$

$= K_H(6) = 0$, $K_H(3) = \frac{3}{C^3_6} = 0{,}15;$ $K_H(4) = \frac{3}{C^4_6} = 0{,}2;$

Отже, даний код має $d_{min} = 3$ і виявляє всі 1-, 2-, 5-, 6-кратні помилки, 85% трьохкратних і 80% чотирьохкратних.

ТЕМА 4. Лінійні блочні коди та матричне їх представлення

Полем Галуа $GF(2)$ будемо називати множину, що складається з двох елементів 0 та 1 із введеними операціями додавання та множення по модулю 2 над ними:

$$0+0=0;\ 0+1=1;\ 1+0=1;\ 1+1=1\ \rightarrow -1=1 \quad \text{/*-> -1=1*/}$$

$$0*0=0;\ 0*1=0;\ 1*0=0;\ 1*1=1\ \rightarrow 1\wedge -1=1 \quad \text{/*-> 1^-1=1*/}$$

Векторним простором $GF^n(2)$ будемо називати множину всіх n-послідовностей елементів з $GF(2)$ з введеними операціями покомпонентного додавання та множення на скаляр.

Лінійним кодом називається підпростір простору $GF^n(2)$.

Елементи цього підпростору називаються *кодовими словами*.

Сума будь-яких двох кодових слів є кодовим словом. Також кодовим словом є добуток будь-якого кодового слова на елемент поля $GF(2)$. Для будь-якого лінійного коду нульове слово завжди є кодовим словом.

Вага Хеммінга $w(c)$ кодового слова дорівнює числу його ненульових компонентів.

Мінімальною вагою *коду* w^* *є мінімальна вага його ненульового кодового слова.*

Теорема. Для лінійного коду мінімальна відстань d^* знаходиться з рівності:

$$d^* = \min_{c \neq 0} w(c) = w^*.$$

Для побудови лінійного коду, що виправляє помилки кратності t, необхідно знайти лінійний код с мінімальною вагою: $w^* \geq 2t+1$.

Для побудови лінійного коду, що виявляє помилки кратності t_0, необхідно знайти лінійний код с мінімальною вагою, що задовольняє нерівності: $w^* > t_0 + 1$.

Матричне представлення лінійних блочних кодів

Так як лінійний код є векторним простором розмірністю n і потужністю 2k , його можна повністю задати k базисними лінійно-незалежними ненульовими n -розрядними векторами (кодовими словами). При цьому будь-яка кодова комбінація коду може бути представлена у вигляді лінійної комбінації базисних кодових слів.

На кодові слова, які складають базис, для забезпечення необхідної мінімальної кодової відстані коду додатково накладаються такі умови:

1. Кожне з вихідних кодових слів повинно містити не менше d_{min} одиниць.

2. Кодова відстань між будь-якими парами вихідних кодових слів не повинна бути меншою за d_{min}.

Базисні кодові комбінації, що однозначно визначають код, зручно представляти у вигляді матриці G, яка називається породжуючою:

$$G = \begin{matrix} a_{11} & a_{12} & \dots & a_{1k} & b_{11} & b_{12} & \dots & b_{1r} \\ a_{21} & a_{22} & \dots & a_{2k} & b_{21} & b_{22} & \dots & b_{2r} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kk} & b_{k1} & b_{k2} & \dots & b_{kr} \end{matrix}$$

Породжуюча матриця G має розмірність k x n і складається з двох підматриць: інформаційної A і перевірочної B . Число стовпців інформаційної підматриці A рівне k , число стовпців перевірочної матриці рівне r :

$$G = |AB|$$

У випадку матричного задання процес кодування лінійним кодом записується у вигляді рівняння: $c = iG$, де i - інформаційне слово (вектор), що кодується; c – відповідне кодове слово коду.

Приклад. Нехай лінійний код задається утворюючою матрицею:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Знайдемо кодове слово, що відповідає інформаційному слову $i = [0\,11]$:

$$c = [011] \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} = [01110]$$

Оскільки код G є підпростором, він має ортогональне доповнення G^T, що складається з усіх векторів, ортогональних до G. Ортогональне доповнення також є підпростором і, таким чином, може розглядатися як код. Такий код G^T називається *дуальним до* G *кодом*.

Ортогональне доповнення G^T має розмірність $r = n - k$, а значить будь-який його базис складається з $n - k$ векторів.

Нехай H — матриця з рядками, що є цими базисними векторами Тоді n -послідовність c є кодовим словом тоді і тільки тоді, коли вона ортогональна кожному вектору-рядку матриці H, тобто коли

$$cH^T = 0,$$

Ця рівність дозволяє перевірити чи є дане слово кодовим. Матриця Н називається *перевірочною матрицею коду*. Розмірність перевірочної матриці: $(n-k)xn$.

Оскільки остання рівність виконується при підстановці замість с будь-якого рядка матриці G, то справедливою є рівність:

$$GH^T = 0.$$

Для попереднього прикладу перевірочною матрицею є матриця:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Теорема. Матриця коду, що породжує, G є перевірочною матрицею дуального коду G^T.

Зв'язок мінімальної ваги коду і перевірочної матриці встановлюється наступною теоремою:

Теорема. Код G містить ненульове кодове слово ваги Хеммінга не більше w тоді і тільки тоді, коли H містить множину з w лінійно залежних стовпців.

Наслідок Код має мінімальну вагу не менше w тоді і тільки тоді, коли кожна множина з $w-1$ стовпців матриці H є лінійно незалежною.

Два коди G і G' називаються *еквівалентними*, якщо їхні породжуючи матриці, утворюються одна з іншої:

1) перестановкою стовпців;

2) елементарними операціями над рядками матриці.

Будь-яка породжуюча матриця, за допомогою даних перетворень може бути приведена до виду в якому інформаційна підматриця A являє собою одиничну матрицю розмірністю k x k.

Породжуючу матрицю такого виду будемо називати породжуючою *матрицею у систематичному виді*, а відповідний код - *систематичним*.

Для систематичних кодів характерним є те, що перша частина кодового слова довжиною k повністю співпадає з інформаційним словом, що спрощує процедуру декодування коду (прикладами таких кодів є код з перевіркою на парність та код з простим повторенням).

Породжуюча матриця G для систематичних кодів може бути представлена у вигляді: $G = |E_k K|$. Відповідна їй перевірочна матриця представляється у вигляді $H = |-P^T : E_T|$.

Теорема. Кожен лінійний код еквівалентний систематичному лінійному коду.

Розглядаючи коди у систематичному вигляді, можна одержати просту нерівність, що зв'язує параметри кодів. Ця границя є дуже неточною, але бувають випадки, коли вона виявляється корисною.

Нехай заданий деякий систематичний (n, k)-лінійний код з мінімальною відстанню d^*. Наступна теорема накладає обмеження на можливі значення n, k, і d^*:

Теорема (границя Сінглтона). Мінімальна відстань (мінімальна вага) будь-якого лінійного (n, k)-коду задовольняє нерівності $d^* \le 1 \div n - k$.

Код з мінімальною відстанню, що задовольняє рівності

$$d^* = 1 \div n - k$$

називається *кодом з максимальною відстанню*.

Границя Сінглтона показує, що для виправлення t помилок код повинний мати не менш $2t$ перевірочних символів (2 перевірочних символи на помилку) Більшість кодів, навіть оптимальних, мають набагато більше перевірочних символів, чим вимагає границя Сінглтона, але параметри деяких кодів задовольняють границі з рівністю. Коди з максимальною відстанню мають рівно $2t$ перевірочних символів.

Приклад. Побудувати матрицю систематичного коду, здатного виправляти одиночну помилку ($t_i = 1$) при передачі 16 повідомлень (N_p). Так як $N_p = 2^k$, то $k = 4$ ($16=2^4$) і число рядків породжуючої матриці G повинно бути рівним 4. Число стовпців матриці G рівна довжині коду $n : n = k + r$, де r - число перевірочних розрядів.

Для $t_i = 1$ $d_{min} = 2t_I + 1 = 3$. Вибираємо $r = 3$. Отже, число стовпців підматриці P рівне трьом.

Враховуючи, що кількість одиниць в рядку підматриці P повинна бути не меншою $d_{min} - 1 = 3 - 1 = 2$ одиниць, вибираємо слідуючі комбінації: 111,110,101,011.

Кінцевий вигляд:

$$G_1 = \left|\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 7 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 6 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 3 \end{array}\right|$$

$$G_2 = \left|\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 7 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 3 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 5 \end{array}\right|;$$

$$G_3 = \left|\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 5 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 7 \end{array}\right|.$$

Побудуємо, наприклад, з допомогою матриці G_3 всі комбінації даного коду (7,4).

Всі 16 комбінацій коду (7,4) з матрицею G_3 виглядають таким чином:

1. 0000000
2. 1000011
3. 0100101
4. 0010110
5. 0001111
6. 1100110
7. 1010101
8. 1101000
9. 0110011
10. 010101
11. 0011001
12. 1110000
13. 0111100
14. 1011010
15. 1101001
16. 1111111

Приклад. Побудувати перевірочну матрицю H коду (7,4), породжуючи матриця якого має вигляд:

$$G_{7,4} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{vmatrix}$$

Знаходимо спочатку підматрицю P^{T}, яка є транспонованою по відношенню до матриці P:

$$P^T = \begin{vmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{vmatrix}$$

Дописуємо до неї справа одиничну матрицю E_{r} і одержимо перевірочну матрицю

$$H = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{vmatrix}$$

ТЕМА 5. Методи виправлення помилок для блочнихлінійних кодів

Існує два основних методи виправлення помилок для лінійних блочних кодів.

Перший метод базується на використанні кодів-супутників. В цьому випадку будується кодова таблиця, в першому рядку якої розміщуються всі кодові слова V_i Другий рядок таблиці заповнюються векторами, держаними в результаті сумування помодулю два кодових слів першого рядка з вектором e_1, вага якого w = 1, а одиниця розміщена в першому розряді. Третій рядок таблиці – результат додавання по модулю двох кодових слів першого рядка з вектором e_2 , вага якого w = 1, а одиниця міститься в другому розряді. Аналогічно заповнюються наступні рядки до тих пір, доки не будуть просумовані з кодовими словами V_i всі вектори e_i вагою w = 1 і одиницями в кожному з n розрядів. Далі сумуються по модулю 2 всі вектори з все можливими e_i до досягнення $w = t_B$, де вагою w = 1, w=2. Таблиця заповнюється t_B – максимальна кількість помилок, яку здатний виправити код.

Таким чином для кожного кодового слова V_i існує своя група кодів-супутників, розміщених у відповідному стовпчику. Всі коди-супутники робочих кодових комбінацій зберігаються в пам'яті машини, яка виправляє код. У випадку прийому комбінації, яка співпадає з одним із кодів-супутників, вона розшифровується як вихідна робоча комбінація для даного коду-супутника.

Приклад. Визначити коди-супутники для коду, який виправляє одиничну помилку з наступними 4-ма робочими комбінаціями: 01001, 01110, 10010, 10101. Виправити помилку в прийнятому кодовому слові 10111. Оскільки $t_B = 1$,достатньо побудувати коди-супутники, відмінні від вихідних кодових комбінацій на e_i з $w = 1$:

	01001	01110	10010	10101
e_1=00001	01000	01111	10011	10100
e_2=00010	01011	01100	10000	10111
e_3=00100	01101	01010	10110	10001
e_4=01000	00001	00110	11010	11101
e_5=10000	11001	11110	00010	00101

Оскільки прийнята спотворена комбінація 10111 є супутнім кодом для вихідного коду 10101, то робимо висновок, що передавалася комбінація 10101.

Розглянутий метод виправлення помилок потребує великого об'єму пам'яті обладнання, особливо при довгих кодових комбінаціях. Тому на практиці застосовується інший метод виправлення помилок, при якому використовуються співвідношення, записані на основі перевірочної матриці H.

Перевірка кодової комбінації на прийомній стороні виконується шляхом співставлення прийнятих перевірочних розрядів кодової комбінації і перевірочних розрядів, обчислених на основі прийнятих інформаційних. Їх сума по модулю два називається *синдромом*. Характерною особливістю синдрому є те, що він не залежить від виду переданої комбінації, а повністю визначається помилками, які спотворили прийняту комбінацію:

$$(c + e)H^T = cH^T + eH^T = 0 + eH^T = eH^T = s,$$

де e – вектор помилки, s – синдром (вектор довжиною $r = n - k$).

Між комбінацією синдрому і комбінацією помилки, яка його викликала, немає взаємно-однозначної відповідності: одному і тому ж синдрому відповідає 2^k різних комбінацій помилок. Так, наприклад нульовому синдрому відповідає нульова комбінація помилок, а також $2^k - 1$ комбінацій помилок, співпадаючих з дозволеними кодовими комбінаціями (невиявлені помилки). Тільки одна з комбінацій помилок, яка відповідає нульовому синдрому, може бути виправлена кодом. При цьому за кожним синдромом закріплюється така комбінація помилок, поява якої в каналі зв'язку

найбільш можлива.

Якщо код використовується для виправлення помилок, то для декодування заздалегідь повинна бути визначена відповідність між видом синдрому і видом помилки, яка виправляється.

Приклад. Розглянемо код, заданий перевірочною матрицею:

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Побудуємо таблицю синдромів, які відповідають помилкам кратності $w=1$ для даного коду. Нехай помилка, виникла в першому

Табл.1

Комбінація помилки	№ спотвореного розряду	Синдром
1000000	1	011
0100000	2	101
0010000	3	110
0001000	4	111
0000100	5	100
0000010	6	010
0000001	7	001

Нехай, наприклад, в кодовій комбінації 1101001 при передачі вийшла помилка в другому розряді і на вхід декодуючого пристрою поступила комбінація 1001001. Виправимо цю помилку, використовуючи таблицю синдромів. Синдром для даної комбінації рівний $s = eH^T = 101$, якому відповідає помилка в другому розряді. Змінивши 0 в другому розряді отриманої комбінації на 1, отримаємо вихідну кодову комбінацію 1101001.

Для виправлення помилок в якості синдромів можуть використовуватися всі комбінації двійкового коду довжиною r , крім нульової, загальна кількість яких дорівнює 2^r -1 . Тому число розрядів синдрому, а, отже, і число перевірочних символів для виправлення однократних помилок, визначається нерівністю:

$$2^r - 1 \geq n = C_n^1, \quad (1)$$

Для виправлення не тільки одиничних, але й подвійних помилок необхідно, щоб виконувалася нерівність:

$$2^r - 1 \geq C_n^1 + C_n^2 \quad (2)$$

В загальному випадку

$$2^r - 1 \geq C_n^1 + C_n^2 + ... + C_n^{t_B} \quad (3)$$

де t_B – максимальна кратність помилки, яку необхідно виправити. Остання умова співпадає з границею Хеммінга.

Якщо для виправлення одиничної помилки одержати синдроми, які дозволяють однозначно визначити місце помилки в комбінації, дуже просто, то для виправлення подвійних, потрійних помилок, а також для виправлення пакетів помилок побудова синдромів є дуже складною задачею. На сьогоднішній день не існує методу, який би дозволяв побудувати систематичний код для виправлення помилок великої кратності. Добре розробленими є лише методи побудови синдромів для кодів, призначених для виправлення подвійних, потрійних помилок та пакетів помилок довжиною $L \leq 3$.

Розглянемо як приклад код з $n = 29$, який виправляє всі потрійні помилки. Синдроми всіх однократних помилок для даного коду представлені в табл.2. Синдроми подвійних і потрійних помилок можна знайти з таблиці 2 шляхом сумування по модулю два синдромів, в рядках яких виникли помилки. Виправлення помилки за допомогою даної таблиці проводиться наступним чином. Нехай, наприклад, прийняте слово має синдром 0000011111. Шукаємо в табл.2 комбінацію з одного, двох або трьох синдромів, які в сумі дадуть отриманий нами синдром. З таблиці видно, що даний синдром одержується шляхом сумування по модулю два 5-го і 6-го розрядів. Отже, робимо висновок, що спотвореними при передачі є п'ятий та шостий розряди кодової комбінації.

Табл.2

№ розряду коду	Синдром	№ розряду коду	Синдром
1	0000000001	16	0011101101
2	0000000010	17	0011110111
3	0000000100	18	0100000000
4	0000001000	19	0100010111
5	0000001111	20	0100101001
6	0000010000	21	0110111101
7	0000100000	22	1000000000
8	0000110011	23	1000011001
9	0001000000	24	1000101101
10	0001010101	25	1001010010
11	0001101010	26	1010000011
12	0010000000	27	1100100011
13	0010010110	28	1101011111
14	0010110101	29	1111100110
15	0011011011		

Через труднощі побудови синдромів для виправлення багатократних помилок, зазвичай процес їх побудови доручають обчислювальним машинам. Так, наприклад, за допомогою обчислювальних машин Банерджі вдалося побудувати таблицю кодів для виправлення будь-яких двохкратних помилок в кодових комбінаціях довжиною до 29 розрядів.

Приклад синдромів для 30-розрядного коду, який виправляє пакети помилок довжиною $L \leq 3$ наведений в табл.2.

Табл.2

№ розряду коду	Синдром	№ розряду коду	Синдром	№ розряду коду	Синдром
1	00000001	11	00001011	21	00010101
2	00000010	12	00010001	22	00100001
3	00000100	13	01000001	23	01001000
4	00001000	14	00001111	24	10000001
5	00010000	15	00100011	25	00011101
6	00100000	16	01000010	26	01000100
7	00001001	17	00001101	27	10000011
8	00010010	18	01000111	28	00110001
9	00100100	19	01010011	29	00010111
10	01000000	20	10000000	30	10000100

Поняття про оптимальні коди

Надлишкові коди можуть застосовуватися з метою виявлення можливих помилок, їх виправлення виявлених помилок або і того і іншого одночасно. В усіх вказаних випадках завжди бажано досягнути максимальної коректуючої здатності, яка здатна змінюватись в широких межах в залежності від конкретної побудови коду.

Виникає питання про те, яка з різновидностей систематичного (n, k)-коду має найвищу коректуючу здатність.

При деяких значеннях n і k може бути знайдена така вдала побудова коду, для якої попарні відстані між всіма дозволеними ненульовими комбінаціями мало відрізняються від половини максимально можливої ваги коду. При інших значеннях n і k може виявитися, що для деякої малої частини кодових комбінацій з їх загального числа 2^k відстань виявляється суттєво меншою, ніж для більшості інших.

При розгляді характеристик кодів можна виявити, що близькі по надлишку числу розрядів коди різко відрізняються один від одного по своїй коректуючій здатності. Крім того, коректуюча здатність одного і того ж коду може значно змінюватись в залежності від характеру розподілу помилок в каналах зв'язку.

Оптимальним рахується код, який при заданих величинах n і k (чи надлишку r) забезпечує найменшу ймовірність не виявлення помилки.

На жаль, загальний аналітичний метод розрахунку оптимальних кодів ще не знайдений. Для довільних значень n і k загальним методом вибору оптимальних кодів все ще залишається метод перебору всіх можливих (n, k)-кодів. В цьому випадку кожний раз визначається, яке значення ймовірності не виявлення помилки $P_{Н.П.}$ є мінімально досягнутим.

За допомогою обчислювальних машин такі розрахунки по знаходженню оптимальних кодів для каналів зв'язку з незалежними помилками були проведені Слепяном, Фонтейном і Пітерсоном.

ТЕМА 6. Коди Хеммінга

Одним з найбільш поширених систематичних кодів є коди Хеммінга. До них відносяться коди з мінімальною кодовою відстанню $d_{min} = 3$, які виправляють всі одиничні помилки і коди з відстанню $d_{min} = 4$, які виправляють всі одиничні і виявляють всі подвійні помилки. Довжина коду Хеммінга

$$n \le 2^r - 1 \qquad (1)$$

З цієї нерівності одержимо

$$2^r - 1 - r \ge k \qquad (2)$$

Формулу (2) можна звести до наступного вигляду:

$$2^k \le \frac{2^n}{1+n} \qquad (3)$$

Остання нерівність дозволяє визначити довжину коду при заданому числі інформаційних розрядів. В таблиці 1 наведені параметри деяких кодів Хеммінга.

Таблиця 1.

k	r	n	$R = r/n$	d	k	r	n	$R = r/n$	d
4	3	7	0,429	3	4	4	8	0,5	4
11	4	15	0,267	3	11	5	16	0,312	4
26	5	31	0,161	3	26	6	32	0,188	4
57	6	63	0,095	3	57	7	64	0,109	4
120	7	127	0,055	3	120	8	128	0,063	4
247	8	255	0,031	3	247	9	256	0,035	4
502	9	511	0,0177	3	502	10	512	0,0195	4
1013	10	1023	0,0098	3	1013	11	1024	0,0107	4

Характерною особливістю перевірочної матриці коду з $d_{min} = 3$ є те, що її стовпці представляють собою любі ненульові комбінації довжиною r. Наприклад, при $r = 4$ і $n = 15$, тобто для коду (15,11), перевірочна матриця може мати наступний вигляд:

$$H_{15,11} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Перестановкою стовпців, які містять одну одиницю, дану матрицю можна звести до вигляду

$$H_{15,11} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Використання такого коду дозволяє виправити будь-яку одиничну помилку чи виявити будь-яку помилку кратності два.

Якщо інформаційні і перевірочні розряди коду нумерувати зліва направо, то у відповідності з матрицею одержимо систему перевірочних рівнянь, з допомогою яких обчислюємо перевірочні розряди:

$$\begin{aligned} U_{12} &= U_5 + U_6 + U_7 + U_8 + U_9 + U_{10} + U_{11}; \\ U_{13} &= U_2 + U_3 + U_4 + U_8 + U_9 + U_{10} + U_{11}; \\ U_{14} &= U_1 + U_3 + U_4 + U_6 + U_7 + U_{10} + U_{11}; \\ U_{15} &= U_1 + U_2 + U_4 + U_5 + U_7 + U_9 + U_{11}; \end{aligned} \qquad (4)$$

де $U_{12} - U_{15}$ – перевірочні розряди; $U_1 - U_{11}$ - інформаційні розряди.

У випадку, коли при передачі кодового слова виникає одинична помилка, виявляться невиконаними ті перевірочні співвідношення, в які входить значення помилкового розряду. Наприклад, коли помилка виникла в п’ятому інформаційному розряді,

виявляться невиконаними перше і четверте рівняння, тобто синдром рівний 1001 (співпадає з п'ятим стовпчиком матриці H). Звідси одержуємо алгоритм визначення місця одиничної помилки: місце положення стовпця матриці H , який співпадає з обчисленим синдромом, вказує місце помилки. Обчислене значення синдрому обов'язково співпадає з одним із стовпців матриці H , так як в якості стовпців вибрані всі можливі двійкові r -розрядні числа.

Хеммінг запропонував використовувати таке розміщення стовпців перевірочної матриці, щоб номер i і-того стовпця матриці і номер розряду кодової комбінації відповідав двійковому представленню числа i і. В цьому випадку синдром, одержаний з перевірочних рівнянь, є двійковим представленням номера розряду комбінації, в якому вийшла помилка. Для цього перевірочні розряди повинні знаходитися не в кінці кодової комбінації, а на номерах позицій, які виражаються степенем двійки (2^0 ,2^2 ,...,2^{r-1}), так як кожний з них входить тільки в одне з перевірочних рівнянь.

Приклад. Для $r = 3,\ \mathrm{n} = 7$, в якості перевірочної матриці може бути вибрана наступна матриця:

---------- $M = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$

В якості перевірочних розрядів вибираємо перший, другий і четвертий. Щоб закодувати повідомлення 1101, потрібно визначити перевірочні розряди $\mathrm{U}_1\mathrm{U}_2$ та U_4 в кодовій комбінації. З матриці H маємо $\mathrm{U}_1 = \mathrm{U}_3 + \mathrm{U}_5 + \mathrm{U}_7;\ \mathrm{U}_2 = \mathrm{U}_3 + \mathrm{U}_6 + \mathrm{U}_7;\ \ \mathrm{U}_4 = \mathrm{U}_5 + \mathrm{U}_6 + \mathrm{U}_7$ Звідки, $\mathrm{U}_1 = 1,\ \ \mathrm{U}_2 = 0,\ \ \mathrm{U}_4 = 0$ і закодоване повідомлення має вигляд 1010101. Уявимо, що шостий символ прийнятий помилково, тоді буде одержано повідомлення 1010111. Синдром в цьому випадку має вигляд 110, тобто є двійковим представленням числа 6.

Двійковий код Хеммінга з кодовою відстанню $d_{min} = 4$ одержується шляхом додавання до коду Хеммінга з $d_{min} = 3$ одного перевірочного розряду, який представляє собою результат сумуванню по модулю два всіх розрядів кодового шару. Довжина коду при цьому $n = 2^r$ розрядів з яких $(r + 1)$ є перевірочними.

Операція кодування може виконуватись в два етапи. На першому етапі визначається кодова комбінація з використанням матриці H, яка відповідає коду з $d_{min} = 3$, на другому – добавляється один перевірочний розряд, в якому записується результат сумування по модулю два всіх розрядів кодового слова, одержаного на першому етапі.

Операція декодування теж складається з двох етапів. На першому обчислюється синдром, який відповідає коду $d_{min} = 3$, на другому – перевіряється останнє перевірочне співвідношення. Результат виконання цих операцій і відповідні їм висновки наведені в таблиці 2.

Таблиця 2.

Синдром	Додаткові контрольні відношення	Висновки
Не рівний нулю	Виконується	Виникла двійкова помилка
Не рівний нулю	Не виконується	Виникла одинична помилка
Рівний нулю	Виконується	Помилок немає
Рівний нулю	Не виконується	Виникла потрійна чи більш високої кратності, але непарна помилка

Перевірочна матриця для (16,11)-коду з $d_{min} = 4$ має вигляд розширеної матриці $H_{15,11}$:

$$H = \begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & | & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & | & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & | & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & | & 0 \\ - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & | & - \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & | & 1 \end{array}$$

Додаткове перевірочне співвідношення, що вводиться для збільшення мінімальної відстані коду Хеммінга, представимо так:

$$U_{16} = U_1 + U_2 + U_3 + U_4 + U_5 + U_6 + U_7 + U_8 + U_9 + U_{10} +$$

$$+ U_{11} + U_{12} + U_{13} + U_{14} + U_{15}.$$

Враховуючи (4):

$$\begin{gathered} U_{16} = U_1 + U_2 + U_3 + U_4 + U_5 + U_6 + U_7 + U_8 + U_9 + U_{10} + U_{11} + U_5 + \\ + U_6 + U_7 + U_8 + U_9 + U_{10} + U_{11} + U_2 + U_3 + U_4 + U_8 + U_9 + U_{10} + U_{11} + \\ + U_1 + U_3 + U_4 + U_6 + U_7 + U_{10} + U_{11} + U_1 + U_2 + U_4 + \\ + U_5 + U_7 + U_9 + U_{11} = U_1 + U_2 + U_3 + U_5 + U_6 + U_8 + U_{11}. \end{gathered}$$

Тому, перевірочна матриця систематичного коду Хеммінга (16,11) буде мати вигляд:

$$H_{16,11} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Інколи при практичному використанні коду Хеммінга зустрічається задача побудови скороченого коду з заданою мінімальною відстанню ($d_{min} = 3$ чи 4). В цьому випадку будується перевірочна матриця для нескороченого коду з найменшим значенням r, яке задовольняє наступним умовам:

$$2^r - 1 - r \geq k \quad \text{для коду з } d_{min} = 3; \quad (5)$$

$$2^r - 1 - r \geq k \text{ для коду з } d_{min} = 4. \quad (6)$$

Після чого в одержаній матриці відкидаються всі лишні стовпці лівої підматриці A матриці H.

Приклад. Побудувати перевірочну матрицю для коду Хеммінга з $d_{min} = 4$, яка містить $\mathrm{k} = 10$ інформаційних розрядів.

Найменше значення r, яке задовольняє нерівність (6), рівно 4. Тобто довжина нескороченого коду рівна 16. Матриця для цього коду рівна:

$$H_{16,11} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Підматриця A перевірочної матриці H, містить 11 розрядів. Тобто, необхідно викреслити з неї один стовпчик. Викреслюємо останній стовпчик для економії апаратури декодування (зазвичай викреслюються стовпчики з найбільшою кількістю одиниць).

Тоді перевірочна матриця скороченого виду коду Хеммінга (15,10) матиме вигляд:

$$H_{15,10} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Для побудови скорочених кодів Хеммінга зручно користуватись табл.1, звідки можна визначити параметри найближчого нескороченого коду. Наприклад, для побудови коду з $\mathrm{k} = 30$ і $d_{min} = 3$, необхідно записати перевірочну матрицю для коду

з $k = 57$ і $d_{min} = 3$, тобто коду (63,57), і викреслити з підматриці $A_{57,6}$ двадцять сім стовпців.

Вагові характеристики коду Хеммінга зручно знаходити з допомогою функції f(x), яка дозволяє визначити число кодових векторів з різними вагами.

Для коду з $d_{min} = 3$:

$$f(x) = 1/(1+n)[(1+x)^n + n(1+x)^{n-1/2}(1-x)^{n+1/2}]$$

Для коду з $d_{min} = 4$:

$$f(x) = 1/(2+n)[(1+x)^n + n(1+x)^{n-1/2}(1-x)^{n+1/2}]$$

В останніх формулах число кодових векторів вагою w рівна значенню коефіцієнтів многочлена, які стоять перед x^w.

Приклад. Нехай заданий код Хеммінга з $d_{min} = 3$, $n = 7$. Визначити коефіцієнти невірних переходів.

Для цього коду маємо $f(x) = 1/8[(1+x)^7 + 7(1+x)^3(1-x)^4] = 1 + 7x^3 + 7x^4 + x^7$.

Отже, даний код має одне кодове слово вагою 0, сім слів вагою 3, сім слів вагою 4 і одне слово вагою 7. Це значить, що розподіл робочих векторів по кодових відстанях для даного коду наступний:

$$N_P(1) = N_P(2) = N_P(5) = N_P(6) = 0,$$
$$N_P(3) = 7,\ N_P(4) = 7,\ N_P(7) = 1.$$

Тоді коефіцієнти невірних переходів:

$$K_{НП}(1) = K_{НП}(2) = K_{НП}(5) = K_{НП}(6) = 0, \qquad K_{НП}(3) = 7 / C_7^3 = 0{,}2;$$
$$K_{НП}(4) = 7 / C_7^4 = 0{,}2; \quad K_{НП}(7) = 1 / C_7^7 = 1.$$

Тобто код виявляє всі одно-, двох-, п'яти-, та шестикратні помилки, 80% трьохкратних і чотирьохкратних.

ТЕМА 7. Циклічні коди

Циклічні коди одержали досить широке застосування завдяки їх ефективності при виявленні і виправленні помилок. Кодери та декодери для цих кодів надзвичайно прості і будуються на основі звичайних регістрів зсуву. Ці коди відносяться до класу лінійних блокових. Назву коди отримали від своєї властивості, яка полягає в тому, що будь-яка кодова комбінація може бути отримана шляхом циклічної перестановки символів комбінації, що належить до цього ж коду. Це означає, що якщо, наприклад, комбінація $a_0a_1a_2...a_{n-1}$ є дозволеною комбінацією циклічного коду, то комбінація $a_{n-1}a_0a_1a_2...a_{n-2}$ також належить цьому коду.

Представлення двійкового коду у вигляді полінома. Операції над поліномами

Циклічні коди зручно розглядати, представляючи комбінацію двійкового коду не у вигляді послідовностей нулів і одиниць, а у вигляді полінома від фіктивної змінної x, а саме:

$$b(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1x + a_0,$$

де a_i - цифри даної системи числення (для двійкової системи – це 0 і 1), знак "+" - сума по модулю два.

Так, наприклад, двійкове семирозрядне число 1110101 може бути записане у вигляді полінома:

$$b(x) = 1*x^6 + 1*x^5 + 1*x^4 + 0*x^3 + 1*x^2 + 0*x + 1.$$

Найбільший ступінь *x*, якому відповідає ненульовий коефіцієнт називається ступенем полінома. У розглянутому прикладі ступінь $b(x)$ дорівнює $degb(x) = 6$.

Представлення кодових комбінацій у вигляді полінома дозволяє звести дії над комбінаціями до дій над многочленами. При цьому додавання двійкових поліномів зводиться до додавання по модулю два коефіцієнтів при рівних ступенях змінної x.

Приклад. $a(x) = x^5 + x^4 + x^3 + 1;\ b(x) = x^7 + x^4 + x^3 + x^2;$

$$a(x) + b(x) = x^5 + x^4 + x^3 + 1 + x^7 + x^4 + x^3 + x^2 = x^7 + x^5 + x^2 + 1.$$

Множення проводиться за звичайним правилом перемножування степеневих функцій, однак, отримані при цьому коефіцієнти при рівних степенях змінної х складаються по модулю два.

Приклад. $a(x) = x^5 + x^2 + 1;\ b(x) = x^3 + x + 1;$

$$a(x) * b(x) = (x^5 + x^2 + 1) * (x^3 + x + 1) = x^8 + x^6 + x^2 + x + 1.$$

Ділення проводиться за правилами ділення степеневих функцій, при цьому операції віднімання заміняються операціями сумування по модулі два.

Приклад. $a(x) = x^5 + x^2 + 1;\ b(x) = x^3 + x + 1;$

$a(x) / b(x) = x^2 + 1 + x / (x^3 + x + 1)$:

$x^5 + x^2 + 1$ ¦ $x^3 + x + 1$

\+ ----------

$x^5 + x^3 + x^2$ ¦ $x^2 + 1$

$x^3 + 1$

\+

$x^3 + x + 1$

----

x - залишок

Побудова циклічних кодів

Ідея побудови циклічних кодів базується на використанні двійкових чисел – многочленів, які є незвідними.

Незвідними називаються многочлени, що не можуть бути представлені у вигляді добутку многочленів нижчих степенів. Вони також, як і прості числа, не можуть бути представлені у вигляді добутку інших чисел. Іншими словами, незвідні поліноми діляться без залишку тільки на себе чи на одиницю.

Приклади незвідних многочленів: $x^3 + x^2 + 1$; $x^4 + x^3 + x^2 + 1$; $x^5 + x^4 + x^2 + 1$; $x^6 + x^5 + x^4 + x^3 + 1$.

Ідея корекції помилок у циклічних кодах базується на тому, що дозволені комбінації коду діляться без залишку на деякий утворюючий поліном, що вибирається з числа незвідних поліномів. Такі поліноми варто шукати серед непарних багаточленів, тобто серед поліномів, що містять непарне число одиниць, тому що з усіх парних

поліномів легко виділити двочлен $(x + 1)$.

Існує два основних способи побудови циклічних кодів.

1-й спосіб

Кодова комбінація циклічного (n, k)-коду виходить шляхом множення простої кодової комбінації степеня (k -1) на одночлен $x^{n-k} = x^r$ і додавання до цього добутку залишку, отриманого відділення отриманогодобутку на утворюючий поліном $g(x)$ степеня n - k = r . При цьому способі кодування перші k символів отриманої кодової комбінації збігаються з відповідними символами вихідної простої кодової комбінації (код є систематичним).

Нехай потрібно закодувати одну з комбінацій чотиризначного двійкового коду: $a(x) = x^3 + x^2 + 1$, тобто 1101. Нехай k = 3 (код буде виправляти всі однократні помилки). З таблиці незвідних поліномів у якості утворюючого вибираємо поліном $g(x) = x^3 + x + 1$ степені r = 3 . Множимо a(x) на одночлен того ж степеня, що й утворюючий многочлен. Від множення багаточлена на поліном x^r ступінь кожногочлена полінома підвищиться на r , що еквівалентно дописуванню rнулів з боку молодших розрядів полінома. Так як deg g(x) = 3 , інформаційну комбінацію a(x) множимо на x^3 :

$$a(x) * x^3 = (x^3 + x^2 + 1) * x^3 = x^6 + x^5 + x^3 = 1101000.$$

Ця процедура здійснюється для того, щоб в результаті замість цих нулів можна було записати контрольні розряди. Значення

контрольних розрядів знаходяться в результаті ділення $a(x)*x^r$ на $g(x)$: $(a(x)*x^3)/g(x) = x^3 + x^2 + x + 1 + 1/(x^3 + x + 1)$, або в загальному вигляді $(a(x)*x^r)/g(x) = Q(x) + R(x)/g(x)$, де $Q(x)$ - частка, а $R(x)$ - залишок від ділення $a(x)$ на $g(x)$.

Останній вираз можна переписати в наступному виді:

$$a(x)*x^r = Q(x)*g(x) + R(x), \text{ або}$$

$$F(x) = Q(x)*g(x) = a(x)*x^r + R(x).$$

У нашому випадку:

$F(x) = (x^3 + x^2 + x + 1)(x^3 + x + 1) = (x^3 + x^2 + 1)*x^3 + 1$, або в двійковому представленні: $F(x) = 1111*1011 = 1101000 + 001 = 1101001$.

Поліном 1101001 і є шукана кодова комбінація, де 1101 - інформаційна частина, а 001 - контрольні символи. Зазначимо, що цей поліном ділиться на утворюючий поліном $g(x)$ без залишку. Звідси отримуємо простий критерій виявлення помилок для циклічних кодів: якщо залишок ділення поліному на утворюючий поліном дорівнює нулю - помилок немає, якщо залишок не дорівнює нулю - помилки є.

2-й спосіб

Кодова комбінація циклічного (n, k)-коду знаходиться шляхом множення простої кодової комбінації ступеня (k -1) на утворюючий поліном g(x) ступеня несистематичний код. n - k = r. При цьому отримується

Нехай потрібно закодувати одну з комбінацій чотиризначного двійкового коду:

$a(x) = x^3 + x^2 + 1$, тобто 1101. За утворюючий поліном виберемо той самий, що і у попередньому прикладі побудови циклічного коду: $g(x) = x^3 + x + 1$. Шукана кодова комбінація в цьому випадку визначається як добуток поліномів $a(x)$ і $g(x)$:

$$(x^3 + x^2 + 1)*(x^3 + x + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$

Отже, шукана кодова комбінація в даному прикладі має вигляд: F(x) = a(x) * g(x) = $x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$, або в двійковому представленні 1111111.

Як і при першому способі, по залишку від ділення кодової комбінації на утворюючий поліном судять про наявність у ній помилок: якщо залишок дорівнює нулю, помилок немає; у протилежному випадку помилки мають місце.

Так як при другому способі в отриманій кодовій комбінації інформаційні символи не завжди збігаються з відповідними символами вихідної простої кодової комбінації, у декодері повинно бути передбачене виділення вихідних символів. Виділення вихідних символів проводиться шляхом поділу кодової комбінації на утворюючий поліном. Наприклад, у розглянутому нами прикладі вихідні символи a(x) визначаються шляхом ділення F(x) на g(x).

Матричне представлення циклічних кодів

Для формування рядків породжуючої матриці, що відповідає 1-му способу утворення циклічного коду, вибираються всі можливі комбінації простого k -розрядного коду a(x) , які містять одиницю в одному розряді. Ці комбінації множаться на x^{nk}, визначається залишок R(x) від ділення отриманого добутку $x^{nk} * ax$ на утворюючий поліном і записується відповідний рядок матриці у вигляді суми добутку $x^{nk} * ax$ залишку R(x) . При цьому породжуюча матриця $P_{n,k}$ представляється двома підматрицями -інформаційною E_k і додатковою $H_r : P_{n,k} \square | E_k, H_r |$.Інформаційна підматриця E_k являє собою квадратну одиничну матрицю з кількістю рядків і стовпців, рівною k . Додаткова підматриця H_r містить r = n - k стовпців і k рядків і утворена залишками R(x).

Породжуюча матриця дозволяє одержати k комбінацій коду. Інші кодові слова виходять сумуванням по модулю два рядків породжуючої матриці у всіх можливих комбінаціях.

Нехай, наприклад, необхідно побудувати породжуючу матрицю $P_{7,4}$ циклічного коду. Породжуючий поліном $g(x) = x^3 + x^2 + 1$.

Інформаційна підматриця має вид:

$$U_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Для отримання першого рядка додаткової підматриці перший рядок інформаційної підматриці множиться на x^3 і ділиться на утворюючий поліном. Залишок цієї операції складає перший рядок додаткової під матриці і дорівнює 110 . Аналогічно визначаються інші рядки додаткової підматриці. Остаточно породжуюча матриця матиме вигляд:

$$P_{7,4} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

При 2-му способі утворення циклічного коду породжуючи Матриця $P_{n,k}$ формується шляхом множення утворюючого полінома g(x) степеня r = n – k на одночлен x^k -1 і слідуючих за цим k -1зсувів триманої комбінації. Інший варіант побудови породжуючої матриці $P_{n,k}$- безпосереднє множення елементів одиничної підматриці на утворюючий многочлен. Незалежно від способу утворення циклічного коду і побудови породжуючої матриці коригуючі властивості коду визначаються тільки обраним утворюючим поліномом g(x) .

ТЕМА 8. Виправлення помилок циклічних кодів
Вибір утворюючого полінома

При побудові циклічного коду спочатку визначається число інформаційних розрядів k по заданому обсягу коду. Потім знаходиться найменша довжина кодових комбінацій n , що забезпечує виявлення чи виправлення помилок заданої кратності. Ця задача зводиться до знаходження потрібного утворюючого полінома $g(x)$.

Оскільки в циклічному коді індикаторами помилок є залишки від ділення полінома прийнятої комбінації на утворюючий поліном, то коректуюча здатність коду буде тим вище, чим більше залишків може бути утворено в результаті цього ділення. Найбільше число залишків, рівне 2^r -1 (крім нульового), може забезпечити незвідний поліном степеня r , який вибирається з спеціальної таблиці незвіднихполіномів:

Табл.1. Таблиця незвідних поліномів степеня ≤ 7

$x^2 + x +1$	$x^6 + x +1$	$x^7 + x^5 + x^2 + x +1$
$x^3 + x +1$	$x^6 + x^3 +1$	$x^7 + x^5 + x^3 + x +1$
$x^3 + x^2 +1$	$x^6 + x^4 + x^2 + x +1$	$x^7 + x^5 + x^4 + x^3 +1$
$x^4 + x +1$	$x^6 + x^4 + x^3 + x +1$	$x^7 + x^5 + x^4 + x^3 + x^2 + x +1$
$x^4 + x^3 +1$	$x^6 + x^5 +1$	$x^7 + x^6 +1$
$x^4 + x^3 + x^2 + x +1$	$x^6 + x^5\ x^2 + x +1$	$x^7 + x^6 + x^3 + x+1$
$x^5 + x^2 +1$	$x^6 + x^5 + x^3 + x^2 +1$	$x^7 + x^6 + x^4 + x +1$
$x^5 + x^3 +1$	$x^6 + x^5 + x^4 + x +1$	$x^7 + x^6 + x^4 + x^2 +1$
$x^5 + x^3 + x^2 + x +1$	$x^6 + x^5 + x^4 + x^2 +1$	$x^7 + x^6 + x^5 + x^2 +1$
$x^5 + x^4 + x^2 + x +1$	$x^7 + x +1$	$x^7 + x^6 + x^5 + x^3 + x^2 + x +1$
$x^5 + x^4 + x^3 + x +1$	$x^7 + x^3 +1$	$x^7 + x^6 + x^5 + x^4 +1$
$x^5 + x^4 + x^3 + x^2 +1$	$x^7 + x^3 + x^2 + x +1$	$x^7 + x^6 + x^5 + x^4 + x^2 + x +1$
	$x^7 + x^4 +1$	$x^7 + x^6 + x^5 + x^4 + x^3 + x^2 +1$

Приклад. Вибрати утворюючий поліном для побудови циклічного коду, що містить k = 4 інформаційних символа і забезпечує твиправлення однократних і виявлення двократних помилок.

Для знаходження числа контрольних символів r можнаскористатися однією з двох формул:

$$r \geq n - k \geq \log_2(n+1),$$

$$r = [\log_2\{(k+1)+[\log_2(k+1)]\}].$$

В нашому випадку за другою формулою $r = [\log_2(5+[\log_2 5])] = 3$ $n = k + r = 7$. А в якості утворюючого поліному g(x) можемо вибрати будь-який поліном степеня 3 з таблиці незвідних, наприклад: $x^3 + x + 1$.

Спряжені многочлени та перевірочна матриця циклічного коду

Поліном q(x) називається спряженим до p(x) поліномом, якщо виконується рівність $q(x) = x^{\deg p(x)} * p(x^{-1})$.

Приклад. Поліноми $p(x) = x^3 + x + 1$ та $q(x) = x^3 + x^2 + 1$ є двоїстими.

Спряжений до деякого незвідного многочлена поліном також є незвідним.

Для знаходження перевірочної матриці циклічного коду зручно використовувати перевірочний поліном h(x) , який знаходиться з утворюючого за формулою:

$$h(x) = (x^n + 1)/p^{-1}(x).$$

Тоді перший рядок перевірочної матриці буде являти собою комбінацію, яка відповідає перевірочному поліному h(x) , до поповнену (r -1) нулями. Другим рядком буде циклічно зсунутий на

один розряд вліво перший, третім – зсунутий другий і т.д.

Приклад. $P(x) + x^3 + x + 1$, $n = 7$. Побудувати перевірочну матрицю.

2. Підтримка кодування окремих областей із кращою якістю. Відомо, що окремі області зображення критичні для сприйняття людиною (наприклад, очі на фотографії), у той час як якістю інших можна знехтувати (наприклад, задній план). При «ручній» оптимізації збільшення ступеня стиску проводиться доти, доки не буде загублена якість у якійсь важливій частині зображення.

3. Основний алгоритм стиску замінений на wavelet. Крім зазначеного підвищення ступеня стиску це дозволило позбавитися від 8-піксельної блочності, що виникає при підвищенні ступеня стиску. Крім того, плавна поява зображення тепер закладена в стандарт (так званий Progressive JPEG, який активно використовується в Інтернет).

4. Для підвищення ступеня стиску в алгоритмі використовується арифметичний стиск.

5. Введена підтримка стиску без втрат. Таким чином, стає можливим використання JPEG для стиску медичних зображень, в поліграфії, при збереженні тексту для розпізнавання OCR-системами і т.п.

6. Підтримка стиску однобітних (2-кольорових) зображень. Для збереження однобітних зображень (малюнки тушшю, відсканований текст і т.п.) раніше завжди рекомендувалися формати GIF та TIFF, оскільки стиск з використанням ДКП дуже неефективний для зображень з різкими переходами кольорів. У JPEG при стисканні 1-бітних зображень, картинка приводилася до 8-бітної, тобто збільшувалася в 8 разів, після чого робилася спроба стиску, нерідко з коефіцієнтом меншим ніж у 8 разів. Зараз можна рекомендувати JPEG 2000 як універсальний алгоритм.

7. На рівні формату підтримується прозорість. Плавно накладати фон при створенні WWW сторінок тепер можна буде не тільки в GIF,

але й у JPEG 2000. Крім того, підтримується не тільки 1 біт прозорості (піксель прозорий/непрозорий), а окремий канал, що дозволяє задавати плавний перехід від непрозорого зображення до прозорого (фону).

ТЕМА 9. Стиск інформації. Основні поняття

Найважливішою характеристикою повідомлення є кількість інформації, яка в ньому міститься. Кількість інформації в повідомленні про подію залежить від імовірності події: чим менш вона ймовірна, тим більше інформації ми одержуємо. Кількість інформації в повідомленні про настання події, ймовірність якої дорівнює P визначається формулою: H =- log (P) .

Одиницею виміру кількості інформації в даному випадку є біт. Таким чином, якщо ми маємо джерело, що видає з однаковою ймовірністю числа 0 і 1 (тобто P(0)= P(1)=1/2), то інформація, що приходиться на одну цифру дорівнює -log(1/2)=1 біт. У випадку російського алфавіту (33 букви) кількість інформації, яка припадає на одну букву при умові однакової ймовірності їх появи в тексті дорівнює log(33)=5.044 бітів, у випадку латинського -log(26)= 4.7, а для десяткових цифр – 3.32 біта на символ.

У випадку, коли цифри з'являються з різною імовірністю, наприклад для алфавіту {0,1}, P(0)= p , P(1)= q =1- p. Тоді кількість інформації, що міститься в одній цифрі буде різною. Якщо P(0) = 0.99 і P(1)= 0.01 , то кількість інформації, що відповідає цифрі "0" буде дорівнювати -log(0.99)= 0.0145 бітів, а цифрі "1" -log(0.01)= 6.64 біта.

Якщо алфавіт складається з N різних символів, що з’являються з ймовірностями p(1),..., p(N) незалежно один від одного. Тоді середня кількість інформації, яка припадає на один символ дорівнює:

H = -p(1)*log(p(1))-...- p(N)*log(p(N)).

Ця величина називається ентропією заданого потоку символів.

Надмірністю ми будемо називати присутність у повідомленні більшої кількості символів, ніж це необхідно

[*максимальна ентропія*] - [*поточна ентропія*] [*Надмірність*] = [*максимальна ентропія*]

Наприклад, у системі з двома символами "ПРО" і "Р", при записі їх як "000" і "111" відповідно надмірність дорівнює (3-1)/3=0.67, тому що 3 двійкові цифри можуть нести 3 біти інформації, а несуть тільки 1.

Всі природні мови мають досить велику надмірність, завдяки чому ми можемо зрозуміти повідомлення, навіть якщо частина букв у ньому зіпсована чи відсутня.

Все сказане вище відноситься до випадку, коли всі символи в повідомленні з'являються незалежно друг від друга. В реальному житті це майже завжди не так. Наприклад у слові "інформа_ія" пропущеною майже напевно є дуже рідка буква "ц". Це означає, що ймовірність появи букви залежить від попередніх (і наступних). У повному обсязі врахувати всі ці залежності дуже складно. На практиці досить гарною моделлю виявляється ланцюг Маркова. У своєму найпростішому вигляді - ланцюг Маркова першого порядку, ми просто вважаємо, що імовірність появи чергової букви залежить лише від попередньої. Щоб задати ланцюг Маркова першого порядку на множині букв латинського алфавіту, нам потрібно задати 26*26- матрицю P , у якій на (i, j)-му місці знаходиться p[i, j]- імовірність появи j -ї букви алфавіту після і -ї. Інколи розглядають ланцюги Маркова і більш високого порядку, у якій імовірність появи чергової букви залежить від декількох попередніх, - двох, трьох і так далі.

Для Марківського ланцюга першого порядку з матрицею p[i, j] кількість інформації середня кількість інформації, яка припадає на один символ (ентропія) дорівнює:

$$H = \underset{i,j}{sum}- p[i, j] * \log_2(p[i, j]) .$$

Остання формула може бути узагальнена і на випадки ланцюгів Маркова вищих порядків. При цьому ентропія з ростом порядку ланцюга Маркова

буде зменшуватись. Наприклад, для англійської мови:

- якщо вважати всі букви рівноймовірними і незалежними, ентропія буде дорівнювати log2 (26)= 4.7 біт/символ.
- якщо вважати, що кожна буква має свою імовірність і букви незалежні одна від одної (Марківській ланцюг нульового порядку), то ентропія буде дорівнює приблизно 4.05 біт/символ.
- при розгляді англійського тексту як Марківського ланцюга другого порядку ентропія виявиться рівною 3.32 біт/символ.
- при розгляді ланцюга третього порядку ентропія дорівнюватиме 3.10 біт/символ.
- реальна же ентропія природної англійського мови, визначена експериментально коливається від 0.6 до 1.3 біта/символ в залежності від тексту.

Отже, "ідеальний" архіватор повинен стискати текстові файли до 8-16% від первісної довжини. На практиці максимальний досягнутий рівень стиску для текстових файлів (архіватор HA) дорівнює 24-30%.

Архіватори

Архіватор - програма, що перетворює деякий двійковий потік даних (файл) в інший меншого або рівного розміру. Відповідно деархіватор - програма, яка виконує зворотне перетворення. Всі алгоритми стиску поділяються на дві групи – алгоритми стиску без втрат і алгоритми стиску з втратами. До першої групи відносяться алгоритми, що точно відновлюють вихідну інформацію. Приклади - lha, arj, pkzip, rar, ha і інші. До другої групи відносяться ті, котрі відновлюють первісну інформацію лише приблизно. Для програм, текстів це є неприпустимим, але для звуку, графіки, відеоінформації це є прийнятним, типовий приклад - алгоритм стиску графічної інформації JPEG, алгоритм стиску звуку MPEG I Layer III, алгоритм стиску відеоінформації MPEG4.

Якби у вхідному потоці всі символи з'являлися з однаковою частотою і були б незалежні друг від друга, то стиснути такий потік символів було б неможливо. Тому не можна побудувати архіватор, який стискав би будь-які файли. При архівації ми повинні використовувати які-небудь особливості вхідного потоку (різну ймовірність появи символів, залежність чергового символу від попередніх).

Стиск послідовності незалежних символів

Будемо вважати, що на вхід нашого алгоритму подається послідовність символів алфавіту A , причому кожний символ a[i] з'являється незалежно від попередніх з ймовірністю p[i]. Тоді вартістю кодування будемо називати середню довжину кодового слова (у бітах). При цьому справедлива наступна теорема:

Теорема (Шеннона). Для будь-якого алгоритму кодування його вартість не нижче ентропії потоку I, де $I= -\log_2 (p[1]) -... \log_2 (p[n])$.

Процес знаходження ймовірностей p[i] появи символів називається моделюванням вхідного потоку. Для моделювання можна використовувати наступні прийоми. По-перше, у деяких випадках ці ймовірності можна вважати даними заздалегідь. Це значно спрощує подальшу роботу і часто дає прекрасні результати, які практично не можна поліпшити. По-друге, можна спочатку переглянути весь вхідний потік і знайти необхідні ймовірності. Необхідність попереднього перегляду дуже важка операція і значно звужує можливість застосування алгоритму. Крім того, може виявитися, що частоти появи символу на початку і наприкінці файлу різні, тому використання однакової частоти символу протягом усього файлу не є оптимальним. Цих недоліків позбавлений адаптивний метод: ймовірністю символу вважається частота його появи серед останніх N символів вхідного потоку. Символам, що узагалі не з’являлися приписується деяка мінімальна ймовірність. На початок роботи алгоритму всі частоти появи символів зазвичай приймаються рівними. В міру перегляду кожного нового вхідного символу вони змінюються, наближаючись до реальних локальних частот

появи символів. І кодер і декодер використовують однакові початкові значення і той самий алгоритм відновлення, що дозволяє їхнім моделям завжди залишатися узгодженими. Кодер одержує черговий символ, кодує його і змінює модель. Декодер визначає черговий символ на основі своєї поточної моделі, а потім обновляє її. При цьому слід враховувати, що постійне відновлення моделі (таблиці частот) потребує великого обсягу пам'яті і процесорного часу. Проте результат найчастіше кращий за два попередньо розглянуті методи, і тому саме цей метод (чи його модифікації) найбільш поширеніші на сьогодні.

ТЕМА 10. Алгоритми стиску інформації без втрат

Код Хаффмена

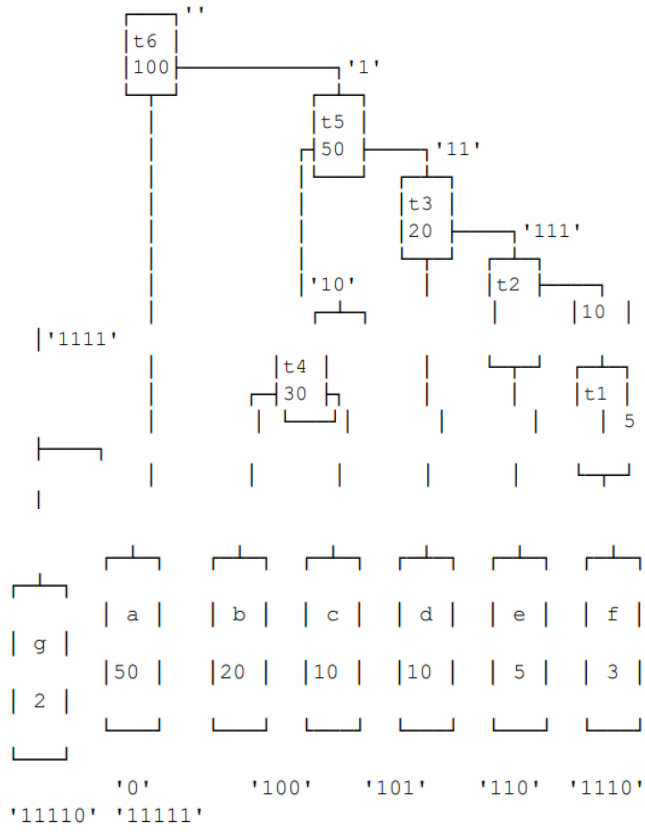
Код Хаффмена зіставляє кожному символу з вхідного алфавіту деякий ланцюжок бітів, довжини яких можуть бути різні. Типовий приклад - абетка Морзе. Нехай символу a[i] зіставляється ланцюжок бітів b[i] . Ясно, що вибір b[i] не може бути довільним. Такий код можна декодувати тільки тоді, коли жоден з ланцюжків b[i] не збігається з початком іншого ланцюжка b[j] (не є префіксом). Код, що задовольняє останній умові називається префіксним. Вартість кодування дорівнює S = p[1]*len(b[1]) +... + p[n] *len(b[n]).

Префіксний код називається оптимальним, якщо його вартість не вище вартості будь-якого іншого префіксного коду. Будь-який оптимальний код для алфавіту A може бути знайдений побудований на основі оптимального коду для меншого алфавіту A_0 . Існує ефективний алгоритм побудови оптимального префіксного коду. Розглянемо цей алгоритм на прикладі.

Приклад. Нехай частоти символів (у відсотках) задані наступною таблицею:

a	b	c	d	e	f	g
50	20	10	10	5	3	2

Побудуємо оптимальне кодове дерево для цього алфавіту (мал.1). На початку ми маємо 7 окремих вузлів (нижній ряд), ніяк не зв'язаних один з одним. На кожному кроці ми додаємо до графа новий вузол, у якого буде рівно 2 нащадки - дві самі рідкі вершини у вже наявному графі. Імовірність нової вершини дорівнює сумі імовірностей нащадків. Процес додавання нових вершин (t1, t2,..., t6) продовжується доти, поки ми не одержимо дерево (вершина - t6). Отримане дерево називається деревом Фано і задає оптимальний префіксний код.



Вартість кодування дорівнює

$$0.5*1+0.2*3+0.1*3+0.1*3+0.05*4+0.03*5+0.02*5 = 2.15,$$

а ентропія потоку - 2.11. Результат кодування методом Хаффмена досить близький до теоретичної межі.

Достоїнствами методу Хаффмена є його досить висока швидкість і добра якість стиску. Цей алгоритм порівняно давно відомий і широко застосовується; прикладами можуть служити програма compress ОС UNIX (програмна реалізація) і стандарт кодування для факсів (апаратна реалізація). Код Хаффмена є оптимальним для кодів, що кодують кожен символ окремим ланцюжком в алфавіті {0,1}.

Недоліком кодування Хаффмена є залежність коефіцієнта стиску від близькості імовірностей символів до негативних ступенів 2; це зв'язано з тим, що кожний символ кодується цілим числом біт. Найбільше яскраво це виявляється при кодуванні двосимвольного алфавіту: у цьому випадку стиск завжди відсутній, незважаючи на різницю в ймовірностях символів; алгоритм фактично "округляє" їх до 1/2. Ця проблема може бути частково вирішена за рахунок блокування вхідного потоку (тобто введення в розгляд нових символів виду 'ab', 'abc',... де a, b, c - символи вихідного алфавіту). Однак це не дозволяє цілком позбутися від втрат (вони лише зменшуються пропорційно розміру блоку) і приводить до різкого росту кодового дерева: якщо, наприклад, символами вхідного алфавіту є 8-бітові байти зі значеннями 0.. 255, то при блокуванні по два символи ми одержуємо 65536 символів (і стільки ж листів кодового дерева), а при блокуванні по три - 16777216! Відповідно зростають вимоги до пам'яті і часу побудови дерева (а при адаптивному кодуванні - і часу відновлення дерева, а виходить, і часу стиску). Втрати ж складуть у середньому 1/2 біта на символ при відсутності блокування, а при його наявності - 1/4 і 1/6 біта відповідно для блоків довжин 2 і 3. Велику степінь стиску, що не залежить від близькості значень ймовірності символів до ступенів 1/2, може дати так зване арифметичне кодування.

Арифметичне кодування

Арифметичне кодування є методом, що дозволяє упаковувати символи вхідного алфавіту без утрат за умови, що відомий розподіл частот цих символів. Арифметичне кодування є оптимальним і досягає теоретичної границі ступеня стиску - ентропії вхідного потоку. При арифметичному кодуванні текст представляється дійсними числами в інтервалі від 0 до 1. По мірі кодування тексту, що стискається, його інтервал зменшується, а кількість бітів для його представлення зростає. Чергові символи тексту скорочують величину інтервалу виходячи зі значень їхніх імовірностей, обумовлених моделлю. Більш ймовірні символи роблять це в меншому ступені, чим менш ймовірні, і, отже, додають менше бітів до результату. Перед початком роботи відповідний тексту інтервал є [0; 1). При обробці чергового символу його ширина звужується за рахунок виділення цьому символу частини інтервалу. Наприклад, застосуємо до тексту "eaii!" алфавіту { a,e,i,o,u,! } модель з постійними імовірностями, заданими в таблиці 1.

Табл.1

Символ	Ймовірність	Інтервал
a	0.2	[0.0; 0.2)
e	0.3	[0.2; 0.5)
i	0.1	[0.5; 0.6)
o	0.2	[0.6; 0.8)
u	0.1	[0.8; 0.9)
!	0.1	[0.9; 1.0)

І кодеру, і декодеру відомо, що на самому початку інтервал є [0; 1). Після перегляду першого символу "e", кодер звужує інтервал до [0.2; 0.5). Другий символ "a" звузить цей новий інтервал до першої його п'ятої частини, оскільки для "a" виділений фіксований інтервал [0.0; 0.2). У результаті одержимо робочий інтервал [0.2; 0.26), тому що попередній

інтервал мав ширину в 0.3 одиниці, а одна п'ята від нього є 0.06. Наступному символу "і" відповідає фіксований інтервал [0.5; 0.6), що стосовно до робочого інтервалу [0.2; 0.26) звужує його до інтервалу [0.23, 0.236). Далі, маємо:

На початку [0.0; 1.0)
"е": [0.2; 0.5)
"а": [0.2; 0.26)
"і": [0.23; 0.236)
"і": [0.233; 0.2336)
"!": [0.23354; 0.2336)

Припустимо, що декодер знає кінцевий інтервал [0.23354; 0.2336) тексту. Він відразу ж знаходить, що перший закодований символ є "е", тому що підсумковий інтервал цілком лежить в інтервалі, виділеному моделлю цьому символу згідно табл.1. Тепер повторимо дії кодера: Спочатку [0.0; 1.0). Після перегляду "е" [0.2; 0.5) Звідси ясно, що другий символ - це "а", оскільки це приведе до інтервалу [0.2; 0.26), що цілком вміщає підсумковий інтервал [0.23354; 0.2336). Продовжуючи працювати в такий же спосіб, декодер знаходить весь текст.

Декодеру немає необхідності знати значення обох границь підсумкового інтервалу, отриманого від кодера. Навіть єдиного значення, що лежить усередині нього, наприклад 0.23355, вже досить. Однак, щоб завершити процес, декодеру потрібно вчасно розпізнати кінець тексту. Крім того, одне і те саме число 0.0 можна представити і як "а", і як "аа", "ааа" і т.д. Для усунення неясності ми повинні позначити завершення кожного тексту спеціальним символом EOT, відомим і кодеру, і декодеру. Для алфавіту з табл.1 для цієї мети, і тільки для неї, використовується символ "!". Коли декодер зустрічає цей символ, він припиняє процес декодування.

Даний метод стиску інформації має недоліки: по-перше, необхідна дійсна арифметика необмеженої точності, і по-друге, результат кодування

стає відомий лише при закінченні вхідного потоку.

Метод RLE (run length encoding)

Суть методу: заміна ланцюжків символів, що повторюються на один цей символ і лічильник повторення. Декодер повинен вміти відрізняти таку кодовану серію від звичайних символів. Тому доводиться до всіх таких ланцюжків додавати деякі заголовки.

Приклад. Виберемо нульовий байт як байт-перемикач. Для кодування одиночного нульового байта будемо використовувати пари < 0,0 >. Для кодування n -байтової послідовності < b,...b > з однакових байтів довжини (4 <= n <= 258) застосуємо ланцюжок довжини 3: < 0,n - 3,b >. Ланцюжок однакових символів довжини 2 і 3 таким методом кодувати безглуздо.

Даний метод буває ефективний для нескладних графічних зображень у форматі "байт на піксель" (наприклад, формат PCX використовує кодування RLE). Недоліки методу RLE очевидні - низький ступінь стиску.

Алгоритм Лемпеля-Зіва

Всі розглянуті вище методи і моделі кодування розглядали в якості вхідних дані ланцюжки символів (тексти) у деякому кінцевому алфавіті. При цьому залишалося відкритим питання про зв'язок цього вхідного алфавіту кодера з тим, що підлягає упаковці (зазвичай також представляється у виді ланцюжків в алфавіті, що складається з 256 символів-байтів).

У найпростішому випадку можна використовувати як вхідний алфавіт кодера саме ці символи (байти) вхідного потоку. Саме так працює метод squashing програми PKPAK (використане статичне кодування

Хаффмена, двупрохідний алгоритм). Ступінь стиску при цьому відносно невелика - біля 50% для текстових файлів.

Набагато більшого ступеня стиску можна домогтися при виділенні з вхідного потоку повторюваних ланцюжків і кодування посилань на них.

Цей метод, про яке і піде далі мова, належить Лемпелю і Зіву, і звичайно називається LZ-compression. Суть його полягає в наступному: кодер постійно зберігає деяку кількість останніх оброблених символів у буфері. По мірі обробки вхідного потоку нові символи попадають у кінець буфера, зсуваючи попередні символи і витісняючи найбільш старі. Розміри цього буфера (ковзного словника - sliding dictionary), варіюються в різних реалізаціях (у LHarc 1.13 використовується 4-кілобайтний буфер, LHA 2.13 і PKZIP 1.10 - 8- кілобайтний, а ARJ 2.20 - 16-кілобайтний).

Алгоритм виділяє (шляхом пошуку в словнику) саму довгу початкову підстроку вхідного потоку, яка збігається з однією з підстрок словника, і видає на вихід пари (length, distance), де length - довжина знайденої в словнику підстроки, а distance - відстань від неї до вхідної підстроки (тобто фактично індекс підстроки в буфері, віднятий від його розміру). У випадку, якщо така підстрока не знайдена, у вихідний потік просто копіюється черговий символ вхідного потоку.

У первісній версії алгоритму пропонувалося використовувати найпростіший пошук по всьому словнику. Час стиску при такій реалізації було пропорційно добутку довжини вхідного потоку на розмір буфера, що є непридатним для практичного використання. Однак, надалі було запропоновано використовувати двійкове дерево для швидкого пошуку в словнику, що дозволило на порядок підняти швидкість роботи алгоритму.

Таким чином, алгоритм Лемпеля-Зіва перетворює один потік вихідних символів у два потоки length і distance. Очевидно, що ці потоки є потоками символів у нових алфавітах L і D , і до них можна застосувати один зі згадуваних вище методів (RLE, кодування Хаффмена, арифметичне кодування). Так ми приходимо до схеми двоступінчастого кодування, найбільш ефективного з сучасних.

Алгоритм LZW

Одним з найпоширеніших методів стиску є алгоритм Лемпеля□Зіва-Велча (Lempel-Ziv-Welch compression, LZW). Алгоритм відрізняють висока швидкість роботи як при упаковці, так і при розпаковці, досить скромні вимоги до пам'яті і простота реалізації. Недолік - низький ступінь стиску в порівнянні зі схемою двоступінчастого кодування.

Метод LZW застосовується в багатьох архіваторах, у графічних файлах формату GIF, для збереження змішаної текстово□графічної інформації в pdf-файлах і в багатьох інших випадках.

Алгоритм LZW не є стандартом - це лише математичний метод, кожна реалізація відрізняється від інших деякими деталями.

Коротко опишемо алгоритм. Самою складною його частиною є словник, що зберігає до декількох тисяч рядків. Над словником можна робити 2 операції: пошук рядка і додавання нового рядка. Додавати можна не довільний рядок, а лише такий, який виходить з деякого іншого, що вже мається в словнику додаванням одного символу (у кінець рядка). Звідси випливає важлива властивість словника, яка широко використовується для оптимізації його роботи: якщо деякий рядок довжини >1 міститься в словнику, то будь-який його початковий підрядок так само міститься в ньому. Ця властивість дозволяє зберігати не весь рядок, а тільки один символ (останній у рядку) і посилання не деякий інший вузол словника, що зберігає початок рядка. Для односимвольних рядків це посилання буде порожнім (NIL).

Пошук рядка в словнику можна організувати по-різному, але це позначиться не на результаті роботи, а тільки на швидкості. Процес же додавання рядка вимагає деяких пояснень. Поки в словнику є вільні місця, то додавання нового рядка не викликає питань. Якщо вільних місць ні, то можливі кілька варіантів дій. У класичному варіанті, словник цілком очищається, точніше кажучи в ньому залишаються тільки односимвольні рядки, що видаляти заборонено. У деяких модифікаціях видаляються

лише рядки, довжини не меншої деякої константи. В інших версіях віддаляються тільки "листи" - ті рядки в словнику, що не є підстроками інших, більш довгих рядків.

Алгоритм упаковки працює так. Припустимо, що вхідний алфавіт складається з m різних символів.

1. Ініціалізація словника. У порожній словник містяться всі m різних односимвольних рядків.

2. Рядку P , називаної "префіксом", привласнюємо порожнє значення.

3. Нехай k - черговий символ вхідного потоку.

4. Шукаємо рядок P + k у словнику. Якщо знайшли, то P:= P + k, зсуваємо покажчик вхідного потоку на наступний символ, переходимо до п. 3. Інакше виводимо номер рядка P у словнику у вихідний потік, додаємо рядок P + k у словник, P:= k (рядок довжини 1), зрушуємо покажчик вхідного потоку на length(P) символів, переходимо до п.3

Таким чином, у вихідний потік попадають номера комірок словника. Якщо словник має розмір 2^s , то на виході ми одержимо потік s -бітних значень. На початку роботи алгоритму, а також після ініціалізації, що відбувається при переповненні словника, його розмір значно зменшується. Тому і довжину вихідних даних можна зменшити. Якщо для представлення всіх символів вихідного алфавіту досить s0 біт, то після ініціалізації для представлення вихідних даних теж досить s0 біт. В міру заповнення словника розмір вихідних символів збільшується до s біт. Ще одна проблема, зв'язана з цим – ефективність пошуку. Найбільш природне її рішення зв'язане з різними варіантами хешування. Тому що для кожного рядка, що знаходиться в словнику, кожен його початковий підрядок теж міститься в словнику, а множину рядків словника можна представити у вигляді дерева. Його корінь - порожній рядок (у словнику відсутній). У кожного словникового рядка може бути кілька нащадків - рядків, що є розширення даної на один символ. Цю структуру можна використовувати

для організації ефективного пошуку. Для цього прийдеться в кожній словниковій комірці зберігати додатково ще два посилання на інші словникові комірки - "молодшого нащадку" і "старшого нащадку". Якщо в словниковій комірці немає молодшого ―брата‖ або нащадків, то ці посилання, природно, повинні вказувати в NIL. При цьому помітно ускладнюються процедури вставки рядка в словник і, особливо, видалення. Але подібне збільшення складності цілком виправдується підвищенням швидкості. На виході з алгоритму не всі коди (номера комірок) будуть з'являтися з рівною ймовірністю. Цю властивість можна використовувати для подальшого збільшення коефіцієнту стиску. Для цього всі гнізда в словнику варто упорядкувати по частоті використання, точніше кажучи організувати двозв'язний список. У кожну комірку доведеться додати ще як мінімум чотири поля - частоту появи цієї комірки, її порядковий номер і номера попередньої і наступної комірок в частотному списку. У вихідний потік варто виводити не просто номер чергової словникової статті, а номер статті, у частотному списку. При цьому маленькі номери будуть з'являтися набагато частіше, ніж великі і для стиску такого потоку можна застосувати або метод Хаффмена, або арифметичне кодування. Звичайно ж, динамічно підтримувати таку складну структуру словника непросто. Тому метод застосовується тільки коли визначальним фактором є сила стиску.

Алгоритм "Shrinking"

"Shrinking" являє собою динамічний Ziv-lempel-Welch алгоритм стиску з частковим очищенням. Початковий розмір вихідного коду - 9 бітів, а максимальний - 13. При цьому мається кілька відхилень від стандарту.

По-перше, розмір не збільшується автоматично, коли розмір таблиці перевищить максимальне значення коду. Це відбувається тільки тоді, коли у вихідний потік дійсно видається відповідний код. Коли декодер

зустрічає код '256,1' (десятковий) він збільшує розмір коду, що читається з вхідного потоку на 1 біт.

По-друге, коли таблиця заповнюється цілком, загальне її очищення не робиться. Замість цього кодер видає код '256,2' і очищає всі "листи" у дереві, що представляє LZW-таблицю. Розмір кодових слів при цьому залишається колишнім. Місця, що звільнилися, заповнюються новими рядками, починаючи з менших значень кодів.

Алгоритм "Imploding"

Алгоритм "Imploding" є об'єднанням двох різних алгоритмів. Перший алгоритм стискає повторювані ланцюжки байтів використовуючи ковзний словник. Другий алгоритм стискає потік, отриманий після першого етапу, використовуючи кілька дерев Фано.

Ковзний словник може мати розмір 4Кб чи 8Кб і визначається бітом у заголовку архіву. Дерева Фано зберігаються на початку стиснутого файлу. Зберігатися може як 2, так і 3 різних дерева. Якщо зберігається 3 дерева, то перше з них використовуються для запису літералів, друге - для запису довжин, третє - для зсувів. У деяких випадках, перше дерево відсутнє.

Дерево літералів використовується для запису довільних байтів і містить 256 значень. Це дерево використовується щоб стиснути байти, не стиснуті алгоритмом ковзного словника. Коли це дерево присутнє, мінімальна довжина збігу (Minimum Match Length, MML) для ковзного словника покладається рівною 3, інакше - 2. Дерево довжини використовується для стиску довжини з пари (довжина, зсув) на виході з ковзного словника.

Дерево довжини містить 64 значення, від MML до 63+MML. Дерево зсувів використовується щоб стиску зсуву з пари (довжина, зсув). Максимальний зсув дорівнює 4Кб чи 8Кб в залежності від розміру словника. Це число занадто велике. Тому дерево містить лише 64

значення, від 0 до 63, що представляють старші 6 бітів зсуву. Молодші біти виводяться у вихідний потік у нестиснутому вигляді.

Самі дерева зберігаються в стиснутому форматі. Перший байт є загальною довжиною стиснутого дерева мінус 1. Кожен наступний байт містить інформацію про числа, код яких має визначену довжину. Якщо ми позначимо молодші 4 біти в байті як 'м', а старші як 'h', то байт 'hm' означає, що (h +1) значень кодуються за допомогою (m +1) біта. Наприклад, код a →'0' ; b→'100' ; c→'101' ; d→'110' ; e→'1110' ; f →'11110' ; g →'11111' розглянутий раніше буде представлений такою послідовністю байтів:

0x04 - довжина, яку займає дерево - 1,

0x00 - один символ має довжину 1,

0x22 - три символи мають довжину 3,

0x03 - один символ довжиною 4,

0x14 - два символи довжиною 5.

При цьому байти з 1-го до 4-го можуть з’являтися в довільному порядку.

Алгоритм "Deflating"

Відмінне сполучення простоти й ефективності стиску представляє алгоритм "Deflating", що використовується в архіваторі PKZIP починаючи з версії 1.93. Починаючи з цієї версії він став основним в архіваторі. Через широку поширеність опишемо його докладніше.

Алгоритм являє собою варіант методу LZ77 (Lempel-Ziv 1977). Він знаходить повторно рядки, які зустрічаються у вхідних даних. Друге входження рядка замінюється покажчиком на попереднє входження, у вигляді пари (відстань, довжина). Відстані обмежені 32К байтами а довжина - 258 байтами. Коли не знайдено збігів достатньої довжини, черговий байт видається у вихідний потік у вигляді літералу.

Літерали і символи довжини стискаються кодом Хаффмена, використовуючи одне з готових дерев Фано, а символи відстані стискаються за допомогою іншого дерева. Алфавіт для першого коду складається з усіх байтів 0..255 плюс всі символи довжиною 3..258. У такий спосіб загальна кількість символів в алфавіті дорівнює 512. Загальна кількість можливих відстаней (32Кб) надто велика для того, що б його можна було ефективно обробити кодом Хаффмена. Тому з 15 біт використовуються тільки старші 9, а молодші 6 бітів передаються без змін. Таким чином, у нас отримуються два різних дерева Фано, кожне з 512 вершинами. Слід зазначити, що для коду Хаффмена не потрібно, що б розмір алфавіту був ступенем двійки, він може бути зовсім довільним, ніяких втрат ефективності при цьому не відбувається.

Дерева зберігаються в компактній формі на початку кожного блоку. Блоки можуть мати довільний розмір (але стиснуті дані одного блоку повинні міститися в наявній пам'яті). Блок закінчується коли архіватор вирішує, що буде ефективніше почати новий блок з обновленими деревами. Рядки, що повторюються, знаходяться з використанням хеш-таблиць. Всі вхідні рядки довжиною 3 у неї вставляються.

Хеш-функція обчислюється на основі наступних 3 байтів. Якщо хеш-ланцюг для цього значення хеш-функції не порожній, то всі рядки в ланцюжку порівнюються з поточним вхідним рядком, і серед них вибирається найбільш довгий збіг. Хеш-ланцюги проглядаються починаючи з самих останніх рядків. Це сприяє тому, що частіше зустрічаються маленькі відстані і, таким чином, код Хаффмена на другому етапі алгоритму дає кращий стиск. Хеш-ланцюги являють собою однозв'язний список. Процедура видалення з ланцюжка відсутня, алгоритм пошуку просто відкидає занадто старі входження рядка. Можлива ситуація, коли деякі з хеш-ланцюгів виявляться надто довгими (хоча це і малоймовірно). В результаті робота алгоритму сильно сповільнюється, тому що доводиться на кожному кроці робити порівняння поточної рядка з усіма рядками з хеш□ланцюга. Щоб цього не

відбувалося, вводиться обмеження на довжину хеш-ланцюгів: занадто довгі просто обрізаються.

Крім того застосовується ще один нескладний спосіб поліпшення роботи алгоритму. Після того, як знайдений придатний рядок у хеш-ланцюзі довжиною N , ми намагаємося знайти більш довгий придатний ланцюжок, починаючи з наступного байта вхідного рядка. Якщо більш довгий рядок знайдений, то рядок з попереднього кроку усікається до довжини 1, тобто у вихідний потік виводиться просто літерал. Якщо ж такий рядок не знайдений, то у вихідний потік виводиться пара (відстань, довжина), що відповідає знайденому рядку і пошук продовжується через N байт.

ТЕМА 11. Алгоритми стиску інформації з втратами

Алгоритми стиску інформації з втратами використовуються для стискання мультимедійної інформації (зображень, звуку, відео). Прикладами такого роду алгоритмів є алгоритми стиску зображень JPEG (використовує дискретно-косинусне перетворення), FIF (використовує теорію фракталів), IW44, DjVu, JPEG2000 (використовує Wavelet-перетворення); алгоритми стиску звукової інформації GSM (стільниковий зв'язок), алгоритм MPEG I Layer III - MP3 (використовується в цифрових аудіоплейєрах); алгоритми стискання відеоінформації MPEG, MPEG2, MPEG4, DivX. Розглянемо як приклад один з алгоритмів стиску зображень – JPEG.

Алгоритм JPEG

JPEG — один з найпоширеніших з сучасних алгоритмів стиску зображень. Алгоритм оперує областями 8x8 пікселів, на яких яскравість і колір змінюються порівняно плавно. Внаслідок цього, при розкладанні матриці такої області в подвійний ряд по косинусах великими

виявляються тільки перші коефіцієнти. Таким чином, стиск у JPEG здійснюється за рахунок плавності зміни кольорів у зображенні. Алгоритм розроблений групою експертів в області фотографії спеціально для стиску 24-бітних зображень. JPEG — Joint Photographic Expert Group — підрозділ у рамках ISO — Міжнародної організації по стандартизації. Назва алгоритму читається як ['jei'peg]. В цілому алгоритм заснований на дискретному косинусоїдальному перетворенні (надалі ДКП), яке застосовується до матриці зображення для одержання деякої нової матриці коефіцієнтів. Для одержання вихідного зображення застосовується зворотне перетворення.

ДКП розкладає зображення по амплітудах деяких частот. Таким чином, при перетворенні ми одержуємо матрицю, у якій багато коефіцієнтів або близьких, або рівних нулю. Крім того, завдяки недосконалості людського зору, можна апроксимувати коефіцієнти більш грубо без помітної втрати якості зображення. Для цього використовується квантування коефіцієнтів (quantization). У найпростішому випадку — це арифметичне побітове зрушення вправо. Під час такого перетворення втрачається частина інформації, але досягається великий коефіцієнт її стиску.

Розглянемо алгоритм докладніше. Нехай ми стискаємо 24-бітне зображення.

Крок 1 Переводимо зображення з колірного простору RGB, з компонентами, що відповідають за червону (Red), зелену (Green) і синю (Blue) складові кольору точки, у колірний простір YCrCb (іноді називають YUV). У останньому Y — яскравість, а Cr, Cb — компоненти, що відповідають за колір (хроматичний червоний і хроматичний синій). За рахунок того, що людське око менш чуттєве до кольору, ніж до яскравості, з'являється можливість архівувати масиви для Cr і Cb компонент з великими втратами і, відповідно, великими ступенями стиску. Подібне перетворення вже давно використовується в телебаченні. На сигнали, що відповідають за колір, там виділяється більш вузька смуга

частот.

Спрощено переклад з колірного простору RGB у колірний простір YCrCb можна представити за допомогою матриці переходу:

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}$$

Зворотне перетворення здійснюється множенням вектора YUV на обернену матрицю:

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 0.772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right)$$

Крок 2.

Розбиваємо вихідне зображення на матриці 8x8. Формуємо з кожної три робочі матриці ДКП — по 8 біт окремо для кожної кольорової компоненти. При великих ступенях стиску цей крок може виконуватися трохи складніше. Зображення ділиться по компоненті Y — як і в першому випадку, а для компонент Cr і Cb матриці значення вибираються через рядок і через стовпець. Тобто з вихідної матриці розміром 16x16 виходить тільки одна робоча матриця ДКП. При цьому, втрачається 3/4 корисної інформації про кольорову складову зображення і відразу отримується стиск у два рази. На якість результуючого RGB зображення, як показує практика, це впливає несуттєво.

Крок 3.

Виконується ДКП елементів матриці 8x8 для кожної з компонент зображення окремо. Спрощено ДКП можна представити у вигляді:

$$Y[u,v] = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C(i,u) \times C(j,v) \times y[i,j],$$

де $C(i,u) = A(u) \times \cos\left(\frac{(2xi+1) \times u \times \pi}{2n} \right)$, $A(u) = \begin{cases} \frac{1}{\sqrt{2}}, for\ u \equiv 0 \\ 1, for\ u \neq 0 \end{cases}$.

При цьому ми отримуємо матрицю, в якій коефіцієнти в лівому верхньому куті відповідають низькочастотній складовій зображення, а в правому нижньому — високочастотній. Поняття частоти випливає з розгляду зображення як двовимірного сигналу (аналогічно розгляду звуку як сигналу). Плавна зміна кольору відповідає низькочастотній складовий, а різкі переходи — низькочастотній.

Крок 4

Робимо квантування. Квантування представляє собою ділення робочої матриці на матрицю квантування поелементно. Для кожної компоненти (Y, U і V), в загальному випадку, задається своя матриця квантування q[u, v] (далі МК).

$$Yq[u, v] = \text{IntegerRound}\left(\frac{Y[u, v]}{q[u, v]}\right)$$

На цьому кроці здійснюється керування ступенем стиску, і відбуваються найбільші втрати інформації. Зрозуміло, що, задаючи МК із великими коефіцієнтами, ми одержимо більше нулів і, отже, більший ступінь стиску.

За елементи МК можна, наприклад, взяти значення, згенеровані алгоритмом:

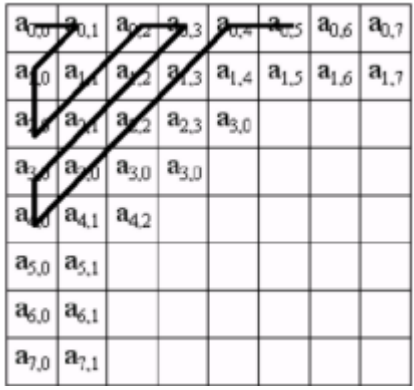
```
for (i=0; i<N; i++)
for (j=0; j<N; J++)
Matrix[i][j]=1+(1+i+j)*QualityFactor
```

В стандарт JPEG включені рекомендовані МК, побудовані експериментальним шляхом. Матриці для більшого чи меншого ступеня стиску одержують шляхом множення вихідної матриці на деяке число gamma. З квантуванням зв'язані і специфічні ефекти алгоритму. При великих значеннях коефіцієнта gamma втрати в низьких частотах можуть бути настільки великі, що зображення розпадеться на квадрати 8x8. Втрати у високих частотах можуть проявитися в так називаному «ефекті Гіббса», коли навколо контурів з різким переходом кольору утворюється

своєрідний «німб».

Крок 5.

Переводимо матрицю 8x8 у 64-елементний вектор за допомогою «зигзаг»-сканування, тобто беремо елементи з індексами (0,0), (0,1), (1,0), (2,0), ...:



Таким чином, на початку вектора ми отримаємо коефіцієнти матриці, що відповідають низьким частотам, а наприкінці — високим.

Крок 6.

Згортаємо вектор за допомогою алгоритму групового кодування RLE. При цьому одержуємо пари типу (пропустити, число), де «пропустити» є лічильником нулів, що пропускаються, а «число»-значення, яких необхідно поставити в наступну комірку. Так, вектор 42 3 0 0 0 -2 0 0 0 0 1 ... буде згорнутий у пари (0,42) (0,3) (3,-2) (4,1) ...

Крок 7.

Згортаємо пари, що вийшли, кодуванням по Хаффману з фіксованою таблицею ймовірностей.

Процес відновлення зображення для алгоритму JPEG цілком симетричний до кодування. Метод дозволяє стискати деякі зображення в

10-15 разів без серйозних втрат якості.

Характеристики алгоритму JPEG

Коефіцієнт стиску: 2-200 (Задається користувачем).

Клас зображень: Повнокольорові 24-бітні зображення або зображення в градаціях сірого без різких переходів кольорів (фотографії).

Симетричність: +

Характерні риси: В деяких випадках, алгоритм створює «ореол» навколо різких горизонтальних і вертикальних границь у зображенні (ефект Гіббса). Крім того, при високому ступені стиску зображення розпадається на блоки 8x8 пікселів.

Алгоритм JPEG 2000

Алгоритм JPEG-2000 розроблений тією же групою експертів в області фотографії, що і JPEG. Формування JPEG як міжнародного стандарту було закінчено в 1992 році. У 1997 стало ясно, що необхідно новий, більш гнучкий і могутній стандарт, що і був дороблений до зими 2000 року. Основні відмінності алгоритму в JPEG 2000 від алгоритму в JPEG полягають у наступному:

1. Краща якість зображення при сильнішому ступені стиску.

2. Підтримка кодування окремих областей із кращою якістю. Відомо, що окремі області зображення критичні для сприйняття людиною (наприклад, очі на фотографії), у той час як якістю інших можна знехтувати (наприклад, задній план). При «ручній» оптимізації збільшення ступеня стиску проводиться доти, доки не буде загублена якість у якійсь важливій частині зображення.

3. Основний алгоритм стиску замінений на wavelet. Крім зазначеного підвищення ступеня стиску це дозволило позбавитися від 8-піксельної блочності, що виникає при підвищенні ступеня стиску. Крім

того, плавна поява зображення тепер закладена в стандарт (так званий Progressive JPEG, який активно використовується в Інтернет).

4. Для підвищення ступеня стиску в алгоритмі використовується арифметичний стиск.

5. Введена підтримка стиску без втрат. Таким чином, стає можливим використання JPEG для стиску медичних зображень, в поліграфії, при збереженні тексту для розпізнавання OCR-системами і т.п.

6. Підтримка стиску однобітних (2-кольорових) зображень. Для збереження однобітних зображень (малюнки тушшю, відсканований текст і т.п.) раніше завжди рекомендувалися формати GIF та TIFF, оскільки стиск з використанням ДКП дуже неефективний для зображень з різкими переходами кольорів. У JPEG при стисканні 1- бітних зображень, картинка приводилася до 8-бітної, тобто збільшувалася в 8 разів, після чого робилася спроба стиску, нерідко з коефіцієнтом меншим ніж у 8 разів. Зараз можна рекомендувати JPEG 2000 як універсальний алгоритм.

7. На рівні формату підтримується прозорість. Плавно накладати фон при створенні WWW сторінок тепер можна буде не тільки в GIF, але й у JPEG 2000. Крім того, підтримується не тільки 1 біт прозорості (піксель прозорий/непрозорий), а окремий канал, що дозволяє задавати плавний перехід від непрозорого зображення до прозорого (фону).

Список літератури

1. Безруков, В.В. Теорія інформації / В.В. Безруков, В.Я. Кізяков, В.І. Профатілов. – Д.: ДДТУЗТ, 2001. – 110 с.
2. Жураковський Ю. П. Теорія інформації та кодування : підручник / Ю. П. Жураковський, В. П. Полторак. – К. : Вища школа, 2001. – 255 с.
3. Коваленко А.Є. Побудова кодів на основі типових алгоритмів кодування даних : методичні вказівки із самостійної роботи студентів з дисципліни «Теорія інформації і кодування» / Уклад. А.Є.Коваленко. Київ: ННК «ІПСА» НТУУ «КПІ», 2012.71 с.
4. Теорія інформації і кодування : Методичні вказівки до проведення практичних занять для студентів напрямів підготовки 6.040302 «Інформатика», 6.040303 «Системний аналіз» / Уклад. А.Є.Коваленко. Київ: ННК «ІПСА» НТУУ «КПІ», 2013. 42 с.
5. Теорія інформації і кодування : Методичні рекомендації до виконання модульної контрольної роботи та залікової роботи для студентів напрямів підготовки 6.040302 «Інформатика», 6.040303 «Системний аналіз / Уклад. А.Є.Коваленко. Київ.: ННК «ІПСА» НТУУ «КПІ», 2013. 22 с.
6. Коваленко А.Є. Побудова кодів на основі типових алгоритмів кодування даних : методичні вказівки із самостійної роботи для студентів з дисципліни «Теорія інформації і кодування» підготовки бакалаврів за спеціальністю “Системний аналіз”2-ге вид., розшир. та доповн. / Уклад. А.Є.Коваленко. Київ.: ІПСА НТУУ «КПІ імені Ігоря Сікорського», 2017. 151 с.
7. Коваленко А.Є. Теорія інформації та кодування: Практикум для студентів напряму підготовки «Системний аналіз». Київ:НТУУ «КПІ», 2014.198 с
8. Кожевников В.Л. Основи збирання, обробки і передачі інформації. Теоретичні основи / В.Л. Кожевников, А.В. Кожевников. – Д.: НГУ,

2007. – 108 с.
9. Майданюк В. П. Кодування та захист інформації. Навчальний посібник. / В. П. Майданюк. - Вінниця: ВНТУ, 2009. - 164 с.
10. Мороз Б.І. Методи раціональної організації обробки інформації в системах передачі даних / Б.І. Мороз, В.П. Дюбко. – Д.: Академія митної служби України, 2007. – 251 с.
11. Подлевський Б. М. Теорія інформації в задачах: підручник / Б. М. Подлевський, Р. Є. Рикалюк. – Київ: «Центр учбової літератури», 2017. – 271 с.
12. Подлевський Б. М. Теорія інформації : підручник / Б. М. Подлевський, Р. Є. Рикалюк. – Львів: Видавничий центр ЛНУ ім. І. Франка, 2016. – 342 с.
13. Тулякова Н. О. Теорія інформації: Навчальний посібник. / Н. О. Тулякова. – Суми: Вид-во СумДУ, 2010. – 248с.

Сінчук Алеся Михайлівна

Теорія інформації та кодування

Курс лекцій

Підписано до друку 28.03.2023 р. Формат 60*84 1/16
Папір друкарський №1. Гарнітура Times. Друк офсетний.
Умовн.-друк.арк. 2,56. Обл.-вид. арк. 2,68.
Тираж 300 прим. Зам. №3.
Віддруковано засобами оперативної поліграфії
редакційно-видавничого відділу
Рівненського державного гуманітарного університету