В. П. Майданюк

КОДУВАННЯ ТА ЗАХИСТ ІНФОРМАЦІЇ

Міністерство освіти і науки України
Вінницький національний технічний університет

В. П. Майданюк

КОДУВАННЯ ТА ЗАХИСТ ІНФОРМАЦІЇ

Затверджено Вченою радою Вінницького національного технічного університету як навчальний посібник для студентів напрямів підготовки 6.050101 – «Комп'ютерні науки» та 6.050103 - «Програмна інженерія» всіх спеціальностей. Протокол № 13 від 3 липня 2008 р.

Вінниця ВНТУ 2009

УДК 621.391 (075)
М 91

Рецензенти:

В. А. Лужецький, доктор технічних наук, професор
В. П. Кожем'яко, доктор технічних наук, професор
В. М. Лисогор, кандидат технічних наук, професор

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Майданюк В. П.
М91 **Кодування та захист інформації.** Навчальний посібник. - Вінниця: ВНТУ, 2009. - 164 с.

В посібнику розглянуто основи теорії інформації та кодування включно з криптографією, а також оглядово питання пов'язані з загрозами інформаційній безпеці, які несе програмне забезпечення комп'ютерних систем. Навчальний посібник призначений для студентів напрямів підготовки 6.050101 – «Комп'ютерні науки» та 6.050103 - «Програмна інженерія» для вивчення дисципліни "Кодування та захист інформації", а також окремих розділів інших споріднених дисциплін.

УДК 621.391 (075)

ЗМІСТ

ВСТУП

Кодування *(coding, encode)* і криптографічний *(cryptographic)* захист інформації є частиною більш широкого поняття – теорії інформації *(information theory)*. Основи цієї науки закладені К. Шенноном, який в 1948 – 1949 роках опублікував дві основні свої роботи – "Математична теорія зв'язку" та "Теорія зв'язку в секретних системах" [1]. Ці роботи не тільки дали поштовх розвитку методів економного та завадостійкого кодування, але і заклали основи сучасної наукової криптології (*criptologic*, з грецької kryptos - таємний, logos - слово) - науки, що займається проблемою захисту інформації шляхом її перетворення.

Навчальний посібник містить чотири основних розділи. В першому розділі розглядаються такі базові питання теорії інформації, як кількісні оцінки інформації, моделі джерел інформації та каналів зв'язку, їх основні характеристики. Другий розділ містить розгляд питань ефективного або економного кодування інформації. В третьому розділі розглянуто основи завадостійкого кодування інформації, наведено опис основних методів побудови корегуючих кодів *(correctings codes)*. Теоретичною основою цих напрямків є теорема К. Шеннона для кодування каналу без завад *(noiseless channel)* та із завадами *(channel with noise)*. Четвертий розділ присвячений вивченню основ наукової криптології. В цьому розділі розглянуто основні поняття та методи криптології, показано її взаємозв'язки з іншими розділами теорії інформації.

Крім того, посібник містить ще один розділ, в якому розглянуто питання пов'язані з місцем і роллю програмного забезпечення *(software)* в системі інформаційної безпеки.

Кожний розділ завершується спеціально підібраними вправами та контрольними питаннями, які допоможуть кращому засвоєнню матеріалу та набуттю практичних навичок у використанні методів теорії інформації в комп'ютерних системах.

Навчальний посібник призначений для студентів напрямів підготовки 6.050101 – «Комп'ютерні науки» та 6.050103 - «Програмна інженерія» і може використовуватись при вивченні дисципліни "Кодування та захист інформації", а також для окремих розділів інших споріднених дисциплін.

1 ІНФОРМАЦІЯ ТА ЇЇ КІЛЬКІСНІ ОЦІНКИ

1.1 Основні поняття

Поняття інформації є центральним моментом кібернетики *(cybernetics)*. Під інформацією в широкому смислі слова розуміють відомості про навколишній світ, які ми отримуємо в результаті взаємодії з ним, пристосовування до нього і зміни його в процесі пристосування. Комітет з термінології академії наук рекомендує таке означення інформації: інформація – це відомості, які є об'єктом зберігання, передачі і перетворення [2].

Необхідно розрізняти поняття інформації та повідомлення *(messages)*. Повідомлення – це форма подання інформації. Наприклад, при телеграфічній передачі повідомленням є текст телеграми. Повідомлення для його передачі за відповідною адресою попередньо перетворюють в сигнал *(signal)*. Під сигналом розуміють фізичну величину, яка змінюється і відображає зміст повідомлення. Сигнал – це матеріальний носій повідомлення. В сучасній техніці знайшли застосування електричні, електромагнітні, світлові та інші сигнали. Всі повідомлення за характером змін за часом можна розділити на неперервні *(continuous)* і дискретні *(discrete)*. Неперервні за часом повідомлення відображаються неперервною функцією часу. Дискретні за часом повідомлення характеризуються тим, що поступають у певні моменти часу і описуються дискретною функцією часу.

Оскільки повідомлення звичайно носять випадковий характер, то неперервні повідомлення описуються випадковою функцією часу, а дискретні як послідовність випадкових подій. Повідомлення також можна розділити на неперервні та дискретні за множиною.

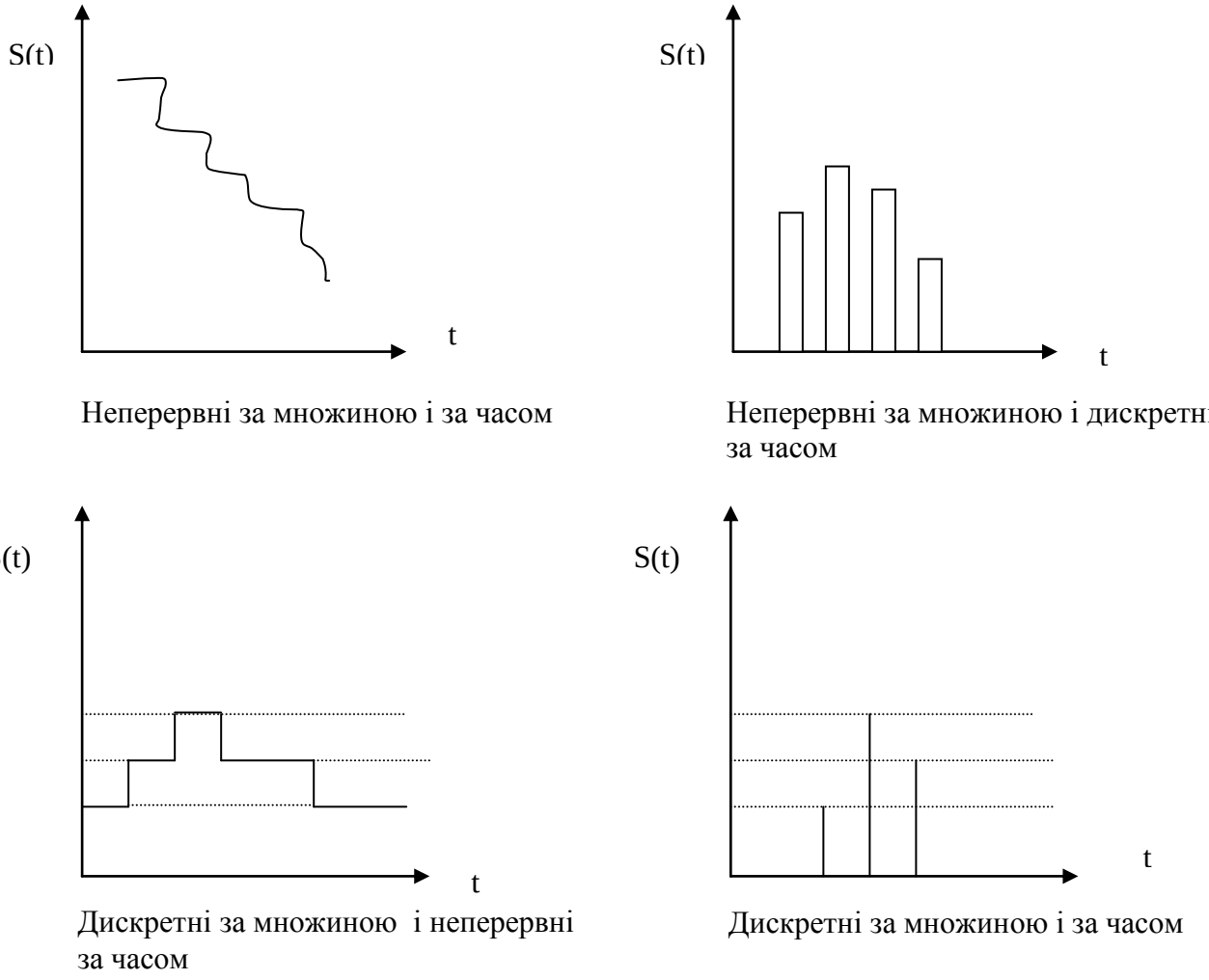
Неперервні за множиною повідомлення характеризуються тим, що функція, яка їх описує, може приймати неперервну множину значень в деякому інтервалі. Дискретні за множиною повідомлення описуються скінченним набором чисел або дискретних значень деякої функції.

Дискретність за множиною і часом не пов'язана одна з одною. Тому можливі такі типи повідомлень[3].

- Неперервні за множиною і часом.
- Неперервні за множиною і дискретні за часом.

- Дискретні за множиною і неперервні за часом.
- Дискретні за множиною і часом.

Графічно ці чотири типи повідомлень подано на рис. 1.1.



Неперервні за множиною і за часом

Неперервні за множиною і дискретні за часом

Дискретні за множиною і неперервні за часом

Дискретні за множиною і за часом

Рисунок 1.1 - Різні типи повідомлень

В широкому смислі перетворення повідомлення в сигнал називають кодуванням повідомлення. В вузькому смислі – кодування – це відображення дискретних повідомлень сигналами у вигляді певних поєднань символів. Пристрій, який виконує кодування, називається кодером *(coder)*, а декодування – декодер *(decoder)*. Кодер і декодер утворюють кодек *(codec)* [2].

1.2 Кількісне оцінювання інформації

Різні способи оцінювання кількості інформації подані в [1-6]. Найбільш простим є комбінаторний *(combinatory)* підхід. Згідно з цим підходом, якщо дискретне джерело інформації U в кожний конкретний момент часу може прийняти випадковим чином один стан із скінченної множини можливих станів N, то ентропія *(entropy)* джерела дорівнює:

$$H(U) = \log N \tag{1.1}$$

Ця міра інформації була запропонована Хартлі в 1928 році. Основа логарифму не має принципового значення і визначає тільки масштаб і одиницю невизначеності. Оскільки сучасна інформаційна техніка базується на двійковій системі числення, то звичайно основу логарифма вибирають рівною двом. При цьому одиниця невизначеності називається двійковою одиницею або бітом *(binary digit - bit)*. Якщо основу логарифма вибрати рівною десяти, то невизначеність отримаємо в десяткових одиницях на один стан - дітах *(dit)*. Основним недоліком комбінаторного підходу є його орієнтованість на системи з рівноймовірними станами, хоча реальні системи не є такими.

Імовірнісний *(probabilistic)* підхід враховує цей недолік. В загальному випадку джерело характеризується сукупністю станів *(states)* з імовірностями *(probabilities)* їх появи (ансамблем).

$$U = \begin{bmatrix} u_1 & u_2 \dots & u_N \\ p_1 & p_2 \cdots & p_N \end{bmatrix} \tag{1.2}$$

$$\Sigma p_i = 1$$

Для джерела інформації з нерівноімовірними станами міра невизначеності була запропонована американським вченим К. Шенноном. Згідно зі Шенноном:

$$H(U) = -C\sum_{i=1}^{N} p_i \log p_i, \tag{1.3}$$

де C – довільне додатне число. Якщо основа логарифма рівна двом, то C = 1, і

$$H(U) = -\sum_{i=1}^{N} p_i \log_2 p_i. \tag{1.4}$$

Запропонована міра названа ентропією, оскільки збігається з ентропією фізичної системи, визначеною раніше Больцманом.

Якщо скористатись умовними імовірностями, то отримаємо умовну ентропію, однак незалежно від способу отримання імовірностей характер залежності (1.4) залишається незмінним.

Алгоритмічний підхід [6] знаходить застосування, якщо дані характеризуються деякими закономірностями, тобто, можуть бути описані деякими формулами або породжувальними (твірними) алгоритмами *(algorithms)*. Тоді ентропія буде рівною мінімальній кількості інформації для передачі цих формул або алгоритмів від джерела до приймача інформації. Алгоритмічний підхід може застосовуватись при ущільненні графічної інформації.

Взаємозв'язок міри Шеннона і Хартлі. Якщо в джерелі може бути реалізовано N рівноймовірних станів, то імовірність кожного з них рівна $\frac{1}{N}$, тоді невизначеність за Хартлі, яка припадає на один стан джерела визначається так [2]:

$$H_i = \log N = \log\frac{1}{p_i} = -\log p_i. \qquad (1.5)$$

Будемо вважати імовірності станів p_i різними, а невизначеність, яка припадає на один стан джерела, за аналогією характеризується величиною:

$$H_i = -\log p_i. \qquad (1.6)$$

Усереднивши по всьому ансамблю U станів джерела знайдемо невизначеність, яка припадає в середньому на один стан джерела:

$$H(U) = -\sum_{i=1}^{N} p_i \log p_i . \qquad (1.7)$$

А це і є міра Шеннона, тобто міра Шеннона є узагальненням міри Хартлі на випадок джерела з нерівноімовірними станами. Вона дозволяє врахувати статистичні властивості джерела інформації [2-3].

Приклад 1.1. Визначити мінімальну кількість зважень, яку необхідно виконати на рівноплечих вагах, щоб серед 27 однакових монет знайти одну фальшиву, яка більш легка.

Розв'язування. Загальна невизначеність ансамблю U відповідно до (1.1) така:

$$H(U) = \log_2 27 = 3 \log_2 3 \text{ біт.}$$

Одне зважування прояснює невизначеність ансамблю U', який нараховує три можливих стани – ліва чаша ваг легша, права чаша ваг легша, ваги знаходяться в рівновазі. Ця невизначеність така: $H(U') = \log_2 3$ біт.

Тобто, $H(U) = 3 H(U')$.

Таким чином, для визначення фальшивої монети достатньо виконати три зважування. Алгоритм визначення фальшивої монети такий. При першому зважуванні на кожну чашу ваг кладуть по дев'ять монет. Фальшива монета буде або на легшій чаші ваг, або серед тих дев'яти, що не зважувались, якщо мала місце рівновага чаш. Аналогічно з дев'яти монет, серед яких знаходиться фальшива, беруть по три монети і кладуть на кожну чашу ваг і виконують друге зважування, яке визначає групу з трьох монет, серед яких одна фальшива. При останньому зважуванні на кожну чашу ваг кладуть по одній монеті, що дає можливість точно указати фальшиву монету.

Приклад 1.2. Порівняти невизначеність, яка припадає на букву джерела інформації (російський алфавіт) з невизначеністю, яка б була у того ж джерела при рівномірному використанні букв.

Розв'язування. Відомо, що кількість букв в російському алфавіті N=32. Тоді при рівномірному використанні букв невизначеність, яка припадає на букву джерела інформації, така:

$$H(U) = \log_2 N = \log_2 32 = 5 \text{ біт.}$$

Але відомо, що букви російського алфавіту в тексті зустрічаються не з однаковою імовірністю [2]. Розподіл імовірностей букв в текстах російською мовою має приблизно такий вигляд:

Таблиця 1.1 - Розподіл імовірностей букв в текстах російською мовою

Буква	Імовірність	Буква	Імовірність	Буква	Імовірність	Буква	Імовірність
А	0,064	й	0,010	т	0,056	ъ, ь	0,015
Б	0,015	к	0,029	у	0,021	ы	0,016
В	0,039	л	0,036	ф	0,02	э	0,003
Г	0,014	м	0,026	х	0,09	ю	0,007
Д	0,026	н	0,056	ц	0,04	я	0,019
Е	0,074	о	0,096	ч	0,013	-	0,0143
Ж	0,008	п	0,024	ш	0,006		
З	0,015	р	0,041	щ	0,003		
И	0,064	с	0,047				

Для джерела інформації із заданим ансамблем згідно з мірою Шеннона (1.7) ентропія джерела така:

$$H(U) = -0,064 \log_2 0,064 - 0,015 \log_2 0,015 - \ldots - 0,143 \log_2 0,143 = 4,42 \text{ біт}$$

Таким чином, нерівномірність імовірностей використання букв зменшує ентропію джерела з 5 до 4,42 біт.

1.3 Властивості ентропії

1. Ентропія –дійсна невід'ємна величина.
2. Ентропія – величина обмежена.
3. Ентропія – рівна нулю, якщо імовірність одного із станів рівна 1, тобто стан джерела повністю визначений.
4. Ентропія – максимальна, коли всі стани джерела рівно імовірні.
5. Ентропія джерела з двома станами u_1 і u_2 змінюються від 0 до 1, досягаючи максимуму при рівності їх імовірностей.
6. Ентропія – об'єднання декількох статистично незалежних джерел інформації дорівнює сумі ентропій початкових джерел. Під об'єднанням двох джерел U і V розуміють узагальнене джерело інформації, яке характеризується імовірностями $p(u_i v_j)$ всіх можливих комбінацій станів u_i джерела U і станів v_j джерела V. Відповідно до означення ентропії [1]

$$H(UV) = -\sum_{i=1}^{N} \sum_{j=1}^{K} p(u_i v_j) \log p(u_i v_j). \qquad (1.8)$$

Оскільки за означенням два джерела статистично незалежні, то

$$p(u_i\, v_j) = p(u_i) * p(v_j).$$

Тоді:

$$H(UV) = -\sum_{i=1}^{N}\sum_{j=1}^{K} p(u_i)\cdot p(v_j)\cdot \log p(u_i)\cdot p(v_j) =$$

$$= -\sum_{i=1}^{N} p(u_i) \log p(u_i) \sum_{j=1}^{K} p(v_j) - \sum_{j=1}^{K} p(v_j) \log p(v_j) \cdot \sum_{i=1}^{N} p(u_i) =$$

$$= -\sum_{i=1}^{N} p(u_i) \log p(u_i) - \sum_{j=1}^{K} p(v_j) \log p(v_j) = H(U) + H(V).$$

Таким чином,

$$H(UV) = H(U) + H(V). \qquad (1.9)$$

Відповідно для ентропії об'єднання декількох джерел маємо:

$$H(UV...Z) = H(U) + H(V) + ... + H(Z). \qquad (1.10)$$

7. Ентропія характеризує середню невизначеність вибору одного стану з ансамблю [2].

1.4 Умовна ентропія і її властивості

При оцінюванні невизначеності часто необхідно врахувати статистичні зв'язки, які в більшості випадків мають місце між станами двох або більше джерел, об'єднаних в рамках однієї системи, або між станами, які послідовно обираються одним джерелом.

Визначимо ентропію об'єднання двох статистично залежних ансамблів U і V.

Об'єднання ансамблів характеризується матрицею P(UV) імовірностей $p(u_i v_j)$ всіх можливих комбінацій станів u_i ансамблю U і станів v_j ансамблю V. Нехай $1 \le i \le N$, $1 \le j \le K$, тоді матриця P(UV) буде мати такий вигляд [2,7]:

$$P(UV) = \begin{pmatrix} p(u_1 v_1) & p(u_2 v_1) & \dots & p(u_N v_1) \\ p(u_1 v_2) & p(u_2 v_2) & \dots & p(u_N v_2) \\ \dots & \dots & & \dots \\ p(u_1 v_K) & p(u_2 v_K) & \dots & p(u_N v_K) \end{pmatrix}. \qquad (1.11)$$

Якщо додати стовпці і рядки матриці, то отримаємо інформацію про ансамблі U і V початкових джерел.

$$U = \begin{pmatrix} u_1 & u_2 & u_N \\ p(u_1) & p(u_2) & p(u_N) \end{pmatrix} \qquad (1.12)$$

$$V = \begin{pmatrix} v_1 & v_2 & v_K \\ p(v_1) & p(v_2) & p(v_K) \end{pmatrix} \qquad (1.13)$$

Відповідно до (1.8):

$$H(UV) = -\sum_{i=1}^{N}\sum_{j=1}^{K} p(u_i v_j) \log p(u_i v_j).$$

З урахуванням того, що $p(u_i v_j) = p(u_i) p_{u_i}(v_j) = p(v_j) p_{v_j}(u_i)$, отримаємо:

$$H(UV) = -\sum_{i=1}^{N}\sum_{j=1}^{K} p(u_i)\cdot p_{u_i}(v_j)\cdot \log p(u_i)\cdot p_{u_i}(v_j) =$$

$$= -\sum_{i=1}^{N}\sum_{j=1}^{K} p(u_i)\cdot p_{u_i}(v_j)\cdot \log p(u_i) - \sum_{i=1}^{N}\sum_{j=1}^{K} p(u_i)\cdot p_{u_i}(v_j)\cdot \log p_{u_i}(v_j) =$$

$$= -\sum_{i=1}^{N} p(u_i)\log p(u_i)\cdot \sum_{j=1}^{K} p_{u_i}(v_j) - \sum_{i=1}^{N} p(u_i)\sum_{j=1}^{K} p_{u_i}(v_j)\log p_{u_i}(v_j)$$

Позначимо $H_U(V) = -\sum_{i=1}^{N} p(u_i)\sum_{j=1}^{K} p_{u_i}(v_j)\log p_{u_i}(v_j)$ - умовна ентропія ансамблю V відносно ансамблю U. Тоді отримаємо:

$$H(UV) = H(U) + H_U(V). \qquad (1.14)$$

Тобто, ентропія об'єднання двох статистично залежних ансамблів (UV) дорівнює сумі ентропій одного з ансамблів і умовній ентропії другого ансамблю відносно першого. Якщо виразити $p(u_iv_j)$ через іншу умовну імовірність, то знайдемо:

$$H(UV) = H(V) + H_V(U). \qquad (1.15)$$

Причому, оскільки $H_U(V) \le H(V)$ і $H_V(U) \le H(U)$, то

$$H(UV) \le H(U) + H(V).$$

Аналогічно, ентропія об'єднання декількох статистично залежних джерел така:

$$H(UVZ\ldots W) = H(U) + H_U(V) + H_{UV}(Z) + \ldots + H_{UVZ\ldots}(W). \qquad (1.16)$$

Приклад 1.3. Визначити ентропію H(U), H(V), $H_V(U)$, H(UV), якщо задана матриця імовірностей станів системи, яка об'єднує джерела U і V:

$$P(VU) = \begin{pmatrix} 0{,}4 & 0{,}1 & 0 \\ 0 & 0{,}2 & 0{,}1 \\ 0 & 0 & 0{,}2 \end{pmatrix}.$$

Розв'язування. Обчислюємо безумовні імовірності станів джерел U і V як суму імовірностей по рядках і стовпцях заданої матриці:

$$p(v_j) = (0{,}4 \;\; 0{,}3 \;\; 0{,}3);$$

$$p(u_i) = \begin{pmatrix} 0{,}5 \\ 0{,}3 \\ 0{,}2 \end{pmatrix}.$$

Тоді

$$H(U) = \sum_{i=1}^{N} p(u_i) \log_2 p(u_i) = -0{,}5 \log_2 0{,}5 - 0{,}3 \log_2 0{,}3 - 0{,}2 \log_2 0{,}2 = 1{,}485 \text{ біт};$$

$$H(V) = \sum_{j=1}^{K} p(v_j) \log_2 p(v_j) = -0{,}4 \log_2 0{,}4 - 0{,}3 \log_2 0{,}3 - 0{,}3 \log_2 0{,}3 = 1{,}57 \text{ біт.}$$

Визначаємо умовні імовірності:

$$p_{v_j}(u_i) = \frac{p(u_i v_j)}{p(v_j)};$$

$$p_{v_1}(u_1) = = \frac{0{,}4}{0{,}4} = 1;$$

$$p_{v_2}(u_1) = p_{v_3}(u_2) = \frac{0{,}1}{0{,}3} = 0{,}33;$$

$$p_{v_2}(u_2) = p_{v_3}(u_3) = \frac{0{,}2}{0{,}3} = 0{,}67;$$

$$p_{v_3}(u_1) = p_{v_1}(u_2) = p_{v_1}(u_3) = p_{v_2}(u_3) = 0.$$

Умовна ентропія ансамблю U відносно до ансамблю V така:

$$H_V(U) = -\sum_j p(v_j) \sum_i p_{v_j}(u_i) \log p_{v_j}(u_i) = -[0{,}4(1 \cdot \log_2 1) + 0{,}3(0{,}33 \cdot \log_2 0{,}33 +$$

$$+ 0{,}67 \cdot \log_2 0{,}67) + 0{,}3(0{,}33 \log_2 0{,}33 + 0{,}67 \log_2 0{,}67)] \approx 0{,}55 \text{ біт}.$$

Ентропія об'єднання двох статистично залежних ансамблів (UV) дорівнює:

$$H(UV) = -\sum_i \sum_j p(u_i v_j) \log_2 p(u_i v_j) =$$

$$= -[0{,}4 \cdot \log_2 0{,}4 + 0{,}1 \cdot \log_2 0{,}1 + 0{,}1 \cdot \log_2 0{,}1 + 0{,}2 \cdot \log_2 0{,}2 + 0{,}2 \cdot \log_2 0{,}2] = 2{,}12 \text{ біт}.$$

Перевіримо результат за формулою:

$$H(UV) = H(V) + H_V(U) = 1{,}57 + 0{,}55 = 2{,}12 \text{ біт}.$$

1.5. Кількість інформації як міра знятої невизначеності

В реальних умовах передача повідомлення по каналах зв'язку *(communication channel)* відбувається при дії завад. Розглянемо дискретне джерело повідомлень *(source of messages)*. Як буде змінюватись невизначеність відносно стану джерела повідомлень при отриманні адресатом елемента повідомлення з виходу каналу зв'язку. Внаслідок дії завад отриманий елемент повідомлення в загальному випадку буде відрізнятись від переданого. При передачі повідомлення $z_1, z_2, ..., z_i, ..., z_N$ отримаємо $\omega_1, \omega_2, ..., \omega_j, ..., \omega_N$.

Вважаючи, що стани джерела реалізуються незалежно, апріорна (*aprior* - до отримання елемента повідомлення) часткова невизначеність появи елемента повідомлення z_i визначається так:

$$H(z_i) = - \log_2 p(z_i), \qquad (1.17)$$

де $p(z_i)$ – апріорна ймовірність появи елемента повідомлення z_i.

Якщо статистичні зв'язки між елементами повідомлення і завадою відсутні, то при отриманні конкретного елемента повідомлення ω_j адресату стає відомою умовна імовірність $p_{\omega_j}(z_i)$, яку називають апостеріорною (*aposterior* - післядослідною) імовірністю реалізації джерелом елемента повідомлення z_i. Тоді апостеріорна часткова невизначеність визначається так:

$$H_{\omega_j}(z_i) = -\log_2 p_{\omega_j}(z_i). \qquad (1.18)$$

А часткову кількість інформації, яку отримують при прийомі елемента ω_j відносно деякого реалізованого джерелом елемента повідомлення z_i, визначимо як різницю часткових невизначеностей до і після отримання елемента повідомлення (апріорної і апостеріорної):

$$I(z_i) = H(z_i) - H_{\omega_j}(z_i) = -\log p(z_i) + \log p_{\omega_j}(z_i) = \log \frac{p_{\omega_j}(z_i)}{p(z_i)}. \qquad (1.19)$$

Якщо завади відсутні, то $p_{\omega_j}(z_i) = 1$, тоді $I(z_i) = - \log p(z_i) = H(z_i)$.

Знайдемо середню кількість інформації, яка міститься в будь-якому прийнятому елементі повідомлення відносно переданого (реалізованого) джерелом.

До отримання конкретного елемента повідомлення середня невизначеність, яка є у адресата відносно реалізації джерелом будь-якого елемента повідомлення, дорівнює ентропії джерела і визначається згідно з (1.7).

Середня невизначеність відносно деякого стану джерела, яка залишається у адресата після отримання конкретного елемента повідомлення ω_j, характеризується частковою умовною ентропією:

$$H_{\omega_j}(Z) = -\sum_{i=1}^{N} p_{\omega_j}(z_i) \log \; p_{\omega_j}(z_i). \qquad (1.20)$$

Це випадкова величина, яка залежить від того, який конкретно елемент повідомлення прийнято.

Середня невизначеність по всьому ансамблю прийнятих елементів повідомлення дорівнює умовній ентропії джерела:

$$H_W(Z) = \sum_{j=1}^{N} p(\omega_j) * H_{\omega_j}(z_i),$$

або

$$H_W(Z) = -\sum_{j=1}^{N}\sum_{i=1}^{N} p(\omega_j) p_{\omega_j}(z_i) \log p_{\omega_j}(z_i). \qquad (1.21)$$

Цю ентропію називають апостеріорною ентропією джерела інформації.

Таким чином, при наявності завад середня кількість інформації, яка міститься в кожному прийнятому елементі повідомлення відносно переданого, дорівнює різниці апріорної і апостеріорної ентропій джерела [2, 7]:

$$I(Z) = H(Z) - H_W(Z). \qquad (1.22)$$

Подавши апріорну і апостеріорну ентропії відповідно до виразів (1.7) і (1.21) отримаємо:

$$I(Z) = -\sum_{i=1}^{N} p(z_i) \log p(z_i) + \sum_{j=1}^{N}\sum_{i=1}^{N} p(\omega_j) p_{\omega_j}(z_i) \log p_{\omega_j}(z_i) =$$
$$= \sum_{j=1}^{N}\sum_{i=1}^{N} p(z_i \omega_j) \log \frac{p(z_i \omega_j)}{p(z_i) p(\omega_j)}. \qquad (1.23)$$

Якщо частковий характер кількості інформації спеціально не указується, то маємо кількість інформації, що припадає в середньому на один елемент повідомлення.

1.6 Модель системи передачі. Кодування джерела інформації та кодування каналу

Інформація від джерела до адресата передається відповідно до структурної схеми, наведеної на рис. 1.2 [2,7-9].

Безпосередня передача сигналу від джерела до отримувача по каналу зв'язку через завади і спотворення часто неможлива, тому повідомлення в кодері попередньо перетворюють в форму більш придатну для передачі по каналах зв'язку. Для спрощення аналізу і синтезу такої структури як кодер,

зручно розділити кодування і декодування, пов’язані з джерелом від кодування і декодування, пов’язаного з каналом. Це спрощує вирішення задач дослідження і синтезу *(synthesis)* кодера для джерела і кодера для каналу. Задачею кодера джерела є забезпечення ефективного кодування повідомлень. Задача кодера для каналу – забезпечення завадостійкості передачі по каналах зв’язку.

Розрізняють джерела дискретних і неперервних повідомлень. Джерело дискретних повідомлень формує дискретні повідомлення із обмеженого числа елементарних повідомлень.

Канал зв’язку – це сукупність пристроїв і фізичних середовищ, які забезпечують передачу повідомлень з одного місця в інше, або від одного моменту часу до іншого.

Якщо канал використовується для передачі дискретних повідомлень, то він називається дискретним каналом, а якщо для передачі неперервних – неперервним.

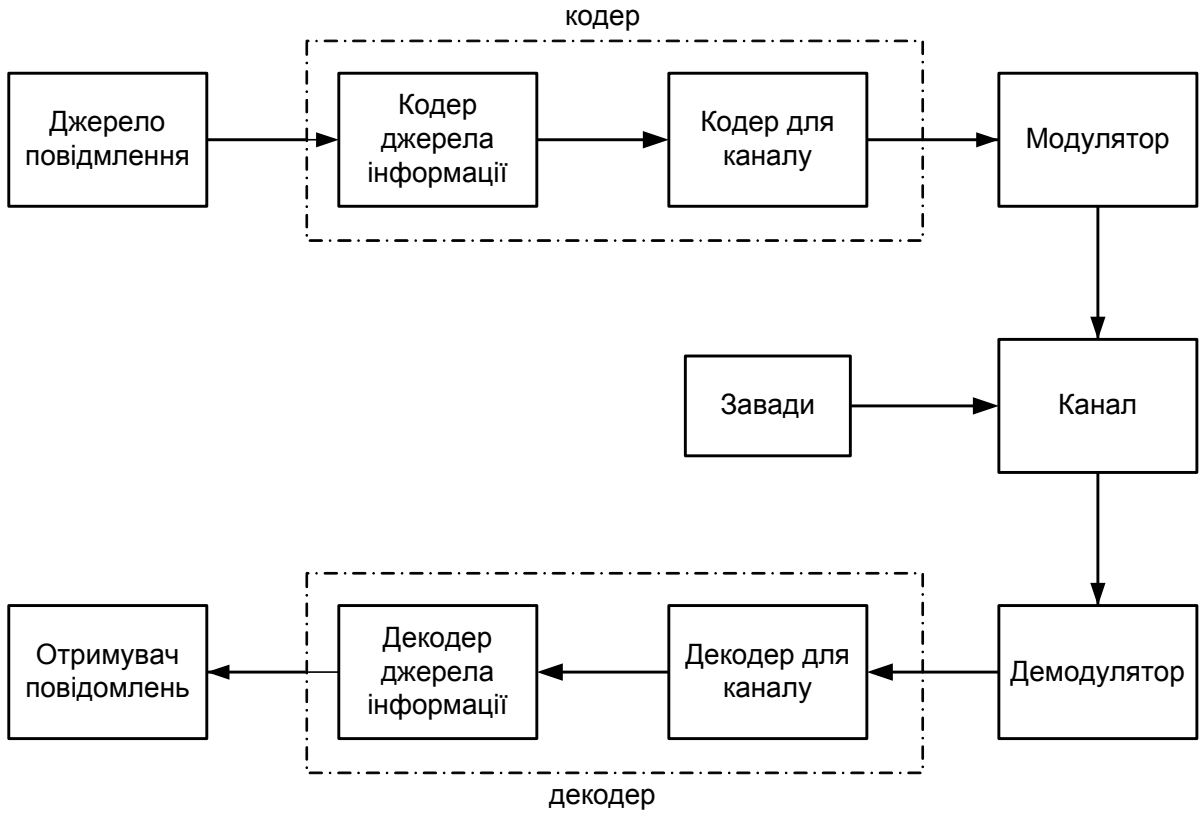


Рисунок 1.2 - Модель системи передачі

Неперервні повідомлення z(t) шляхом дискретизації і квантування завжди можна перетворити в дискретні і таким чином перейти від неперервного каналу до дискретного.

Якщо дією завад в каналі можна знехтувати, то для аналізу використовують модель у вигляді ідеалізованого каналу, який називається каналом без завад, і в якому кожному повідомленню на вході однозначно відповідає повідомлення на виході. Коли вимоги до достовірності великі і знехтувати неоднозначністю між вхідним повідомленням каналу Z і вихідним повідомленням каналу W недопустимо, використовують більш складну модель каналу – канал із завадами.

Канал вважається заданим, якщо відомі статистичні дані про повідомлення на його вході і виході, а також обмеження, які накладаються на вхідне повідомлення фізичними характеристиками каналу.

В подальшому розглядаються лише дискретні повідомлення і канали.

1.7 Модель джерел дискретних повідомлень

Для побудови моделі *(model)* необхідно знати об'єм алфавіту знаків $z_1, z_2,\ldots,z_N$, з яких джерело формує повідомлення і імовірності створення окремих знаків *(characters)* з урахуванням можливих взаємозв'язків між ними. При доведені основних положень теорії інформації К. Шеннон використовував модель, яка називається ергодичним *(ergodic)* джерелом повідомлення. З теорії імовірності відомо, що така модель задовольняє умови стаціонарності *(stationary)* і ергодичності: перше означає, що імовірність окремих знаків і їх поєднань не залежить від розташування останніх по довжині повідомлень, друге означає, що статистичні закономірності, отримані при дослідженні одного достатньо довгого повідомлення з імовірністю близькою до "1", справедливі для всіх повідомлень, що формуються джерелом [1-2].

Стаціонарне джерело повідомлень, яке вибирає кожний знак послідовності, що формується, незалежно від інших знаків, завжди ергодичне і називається джерелом без пам'яті *(source without memory)* або джерело Бернуллі. Якщо всі знаки повідомлення рівноімовірні, то таке джерело без пам'яті називають комбінаторним. А якщо джерело відповідає таким умовам:

$$p(z_i) \geq p(z_j), \quad \text{якщо} \quad i \geq j$$
$$\text{або} \quad p(z_i) \leq p(z_j), \quad \text{якщо} \quad i \leq j, \tag{1.24}$$

то таке джерело називається монотонним.

На практиці, однак, частіше зустрічаються такі джерела, вибір одного знака повідомлення в яких залежить від того, які знаки були вибрані джерелом до того. Такі джерела називаються джерелами з пам'яттю *(source with memory)*. Для опису функціонування таких джерел використовують ланцюги Маркова *(Markov chain)*. Ланцюг Маркова порядку n характеризує послідовність подій, імовірності яких залежать від того, які n подій передували даній. Ці n конкретних події визначають стан джерела, в якому воно знаходиться при формуванні чергового знака.

Нехай L- об'єм алфавіту джерела, тоді число різних станів джерела R не буде перевищувати L^n. Позначимо ці стани як s_1, s_2,…,s_q,…, s_R. А ймовірність вибору в стані s_q знака z_i через $p_q(z_i)$. Якщо джерело знаходиться в стані s_q, то його часткова ентропія буде визначатися так:

$$H(s_q) = -\sum_{i=1}^{L} p_q(z_i) \log p_q(z_i). \tag{1.25}$$

Усереднюючи випадкову величину $H(s_q)$ по всім можливим станам q=1,2,3…R, отримаємо ентропію джерела повідомлення:

$$H(Z) = -\sum_{q=1}^{R} p(s_q) \sum_{i=1}^{L} p_q(z_i) \log p_q(z_i), \tag{1.26}$$

де $p(s_q)$ – імовірність того, що джерело знаходиться в стані s_q. Величина H(Z) характеризує невизначеність, яка в середньому припадає на один знак, який формується джерелом. Розглянемо окремі випадки.

1. Статистичні зв'язки між знаками відсутні, тобто n=0, $R=L^n=L^0=1$, $p(s_1)=1$ і таким чином

$$H(Z) = -\sum_{i=1}^{L} p(z_i) \log p(z_i). \tag{1.27}$$

2. Кореляційні зв'язки спостерігаються тільки між двома знаками (простий ланцюг Маркова). В даному випадку: n=1, R=L. Тоді:

$$p_q(z_i) = p_{z_q}(z_i),$$
$$H(Z) = -\sum_{q=1}^{L} p(z_q) \sum_{i=1}^{L} p_{z_q}(z_i) \log \; p_{z_q}(z_i) \; . \tag{1.28}$$

Аналогічно можна отримати формули і для більш складних випадків, однак на практиці обмежуються значеннями n=1,2, через складність обчислень.

Приклад 1.4. Визначити чи є ергодичним стаціонарне дискретне джерело повідомлень, алфавіт якого складається з 4-х знаків z_1, z_2, z_3, z_4 Причому, безумовні імовірності вибору знаків такі:

$$p(z_1) = p(z_2) = p(z_3) = p(z_4) = \frac{1}{4}.$$

А умовні імовірності задані такою таблицею:

Таблиця 1.2 – Умовні імовірності вибору знаків

$z_q \backslash z_i$	z_1	z_2	z_3	z_4
z_1	1/3	1/3	1/3	0
z_2	1/3	1/3	1/3	0
z_3	1/3	1/3	1/3	0
z_4	0	0	0	1

Розв'язування. Аналіз таблиці показує, що якщо першим буде вибраний один із знаків z_1, z_2, z_3, то джерело почне формувати послідовність з рівноімовірною появою цих знаків, а якщо першим буде вибраний z_4, то формується послідовність, яка містить тільки символи z_4. З урахуванням (1.26) ентропія даного джерела повідомлень складе:

$$H(Z) = -\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{1}{3}\log_2\frac{1}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) - \frac{1}{4}\log_2 1 = 1{,}19 \text{ біт}.$$

Розглянемо два випадки.

1. Першим вибрано один із знаків z_1, z_2, z_3. Тоді ентропія послідовності така:

$$H_1(Z) = \left(-\frac{1}{3}\log_2\frac{1}{3}\right)\cdot 3 = 1.585 \text{ біт}.$$

2. Першим вибрано Z_4. Тоді ентропія послідовності така:

$$H_2(Z) = 1\cdot\log_2 1 = 0.$$

Оскільки ентропії послідовностей не збігаються з ентропією джерела, то воно не є ергодичним.

1.8 Властивості ергодичних послідовностей знаків

Нехай ергодичне джерело без пам'яті формує або послідовно видає знаки z_1, z_2, z_3, з ймовірностями 0,1; 0,3; 0,6, тоді в достатньо довгій послідовності знаків в середньому на один знак z_1 припадатиме три знаки z_2 і шість знаків z_3. Однак при обмеженому числі знаків послідовності існує ймовірність того, що вона міститиме [1-2]:

- тільки знаки z_1 або z_2, або z_3;
- тільки знаки z_1 і один z_2 або z_3;
- тільки знаки z_2 і один z_1 або z_3;
- тільки знаки z_3 і один z_2 або z_1;
- тільки знаки z_1 і два знаки z_2 або z_3;

і т.д.

Із збільшенням кількості знаків N в послідовності, ймовірність появи таких послідовностей зменшується. Фундаментальну властивість довгих послідовностей знаків, які створюються ергодичним джерелом повідомлень відображає така теорема [1-2].

Означення. Які б малі не були два додатні числа $\delta>0$ і $\mu>0$, при достатньо великому N всі послідовності можуть бути розбиті на 2 групи:

- нетипові послідовності
- типові послідовності.

Сумарна ймовірність нетипових послідовностей дуже мала і менша як завгодно малого числа δ. Для типових послідовностей ймовірності їх появи практично однакові і ймовірність p будь-якої такої послідовності задовольняє нерівність:

$$\left| \frac{\log \frac{1}{p}}{N} - H(Z) \right| < \mu, \qquad (1.29)$$

де H(Z) - ентропія джерела повідомлення,

μ - як завгодно мале додатне число.

Це відношення називається властивістю асимптотичної рівномірності довгих послідовностей [2].

Оскільки при $N \to \infty$, джерело повідомлень з імовірністю близькою до “1” видає тільки типові послідовності, кількість яких дорівнює 1/p, а невизначеність кожної послідовності з урахуванням їх рівноімовірності складає log (1/p). Тоді, $(\log \frac{1}{p}) / N$ - невизначеність, яка припадає на один знак повідомлення. Зрозуміло, що ця величина не повинна відрізнятись від ентропії джерела, що і констатує співвідношення (1.29).

Доведемо теорему для найпростішого ергодичного джерела без пам'яті. Нехай є довга послідовність з N елементів. Імовірність появи знаків алфавіту z_1, z_2, ...z_n, дорівнює p_1, p_2, …p_n, і тоді згідно із законом великих чисел в послідовності міститься Np_1 елементів z_1, Np_2 елементів z_2 і т.д. Імовірність p реалізації будь-якої типової послідовності буде близькою:

$$p = p_1^{Np_1} \cdot p_2^{Np_2} \cdot \ldots \cdot p_n^{Np_n} .$$

Логарифмуючи даний вираз отримаємо:

$$\log p = Np_1 \cdot \log p_1 + Np_2 \cdot \log p_2 + \ldots + Np_n \cdot \log p_n = N\sum_{i=1}^{n} P_i \log P_i .$$

Звідки $-\frac{\log p}{N} = -\sum_{i=1}^{n} p_i \log p_i$, або

$$\frac{\log\frac{1}{p}}{N} = H(Z). \qquad (1.30)$$

Що і необхідно було довести.

Покажемо, що за винятком випадку рівноімовірного і незалежного вибору букв джерела, коли нетипові послідовності відсутні, типові послідовності при досить великому N складають незначну частку від загальної кількості послідовностей.

При об'ємі алфавіту джерела n і кількості знаків в послідовності N, загальна кількість можливих послідовностей складає:

$$n_1 = n^N = 2^{\log_2 n^N} = 2^{N\cdot\log_2 n} \qquad (1.31)$$

А число типових послідовностей складе:

$$n_2 = \frac{1}{p} = 2^{N\cdot H(Z)}. \qquad (1.32)$$

Тоді відношення

$$\frac{n_1}{n_2} = 2^{N\cdot(\log_2 n - H(Z)} \qquad (1.33)$$

Оскільки $\log_2 n > H(Z)$, то $n_1 >> n_2$, ця нерівність підсилюється із збільшенням N. Шеннон показав, що розглянуті вище властивості є основою ефективного кодування інформації [1].

Приклад 1.5: Оцінити, яку частку загальної кількості можливих послідовностей необхідно врахувати в практичних розрахунках, якщо ергодичне джерело характеризується такими параметрами: n =16, H(Z) = 3,5 дв.од., N = 50.

Розв'язування. Знайдемо загальну кількість послідовностей:

$$n_1 = n^N = 16^{50} = 2^{200}.$$

Кількість типових послідовностей складе:

$$n_2 = 2^{N \cdot H(Z)} = 2^{50 \cdot 3,5} = 2^{175}.$$

Частка типових послідовностей така:

$$\frac{n_2}{n_1} = \frac{2^{175}}{2^{200}} = \frac{1}{2^{25}} \approx \frac{1}{30 \cdot 10^6}.$$

1.9 Міра надмірності

Міра надмірності *(redundancy)* показує, наскільки добре використовуються знаки даного джерела інформації і визначається так [2]:

$$D = \frac{H_{max}(Z) - H(Z)}{H_{max}(Z)}, \qquad (1.34)$$

де $H_{max}(Z) = \log_2 L$, L – об'єм алфавіту;

H(Z) – ентропія джерела повідомлень.

Якщо надмірність D = 0, то повідомлення, які формуються джерелом повідомлень, оптимальні з точки зору кількості інформації, яку вони переносять. Для передачі певної кількості інформації I при відсутності завад в цьому випадку необхідно:

$$k_1 = \frac{I}{H_{max}(Z)} (\text{знаків}).$$

Оскільки для реального джерела для передачі такої ж кількості інформації I необхідно більше знаків, а саме:

$$k_2 = \frac{I}{H(Z)} > k_1.$$

Тому говорять також про надмірність знаків в повідомленні:

$$D = \frac{k_2 - k_1}{k_2} = \frac{H_{max}(Z) - H(Z)}{H_{max}(Z)}. \qquad (1.35)$$

Наслідки наявності надмірності неоднозначні.

1. Надмірні повідомлення вимагають додаткових витрат на передачу, обробку та зберігання.

2. Наявність надмірності підвищує завадостійкість повідомлення.

Однак в технічних системах надмірність вилучається, а для підвищення завадостійкості вводиться “раціональна” надмірність, яка дозволяє виявити і виправити найбільш імовірні і небезпечні за наслідками помилки простими технічними засобами.

Приклад 1.6. Визначити можливий ефект при передачі тексту українською мовою від вилучення надмірності.

Розв’язування. Відомо, що для тексту українською мовою [2]:

$$H_{max}(Z) = \log_2 32 = 5 \text{ біт.}$$

$$H(Z) = 4{,}42 \text{ біт.}$$

З урахуванням перехідних імовірностей встановлено, що ентропія тексту складає:

$$H(Z) = 1{,}5 \text{ біт.}$$

Тоді надмірність тексту складе:

$$D = \frac{5 - 1.5}{5} = \frac{3.5}{5} = 0{,}7.$$

Тобто, повне вилучення надмірності дозволило б підвищити продуктивність каналу зв’язку майже в 3 рази.

1.10 Продуктивність джерела дискретних повідомлень

Під продуктивністю джерела *(source productivity)* повідомлення розуміють кількість інформації, яку виробляє джерело в одиницю часу [2].

Позначимо тривалість видачі знака z_i, який формується джерелом в стані s_q через τ_{qz_i}. Тоді середня тривалість видачі джерелом одного знака складе:

$$\tau_д = \sum_{q=1}^{R} p(s_q) \sum_{i=1}^{l} p_q(z_i) \tau_{qz_i}. \qquad (1.36)$$

А продуктивність джерела:

$$\bar{I}(Z) = \frac{H(Z)}{\tau_д}. \qquad (1.37)$$

Тривалість знаків бажано вибирати пропорціональною їх імовірностям. Якщо $\tau_0 = \tau$, тобто не залежить від стану джерела, то

$$\bar{I}(Z) = \frac{H(Z)}{\tau}. \qquad (1.38)$$

1.11 Моделі дискретних каналів

Дискретний канал – сукупність засобів, призначених для передачі дискретних сигналів [2]. Дискретні повідомлення, які складаються з послідовності знаків алфавіту z_1, z_2,…,z_n, перетворюються в пристрої кодування в послідовність символів u_1, u_2, u_m. Як правило, m<n, але вони можуть і збігатися. Матеріальне втілення символу – елементарний сигнал, який отримують в процесі маніпуляції, тобто дискретної зміни певного параметра носія інформації.

В результаті маніпуляції кожній послідовності символів ставиться у відповідність складний сигнал. Множина складних сигналів скінченна. Термін елементарний сигнал і символ або складний сигнал і послідовність символів – синоніми. Інформаційна модель каналу із завадами задається множиною символів на його вході і виході та описом імовірних властивостей передачі окремих сигналів. В загальному випадку канал може мати множину станів і переходити з одного стану в інший як з часом, так і в залежності від послідовності символів, що передаються. В кожному стані канал характеризується матрицею умовних імовірностей. $p(v_j/u_i)$ - імовірність того, що переданий символ u_i буде сприйнятий на виході каналу як v_j. Якщо перехідні ймовірності $p(v_j/u_i)$ залежать від часу, що характерно для реальних каналів, то такий канал називається нестаціонарним каналом зв'язку. Якщо ця залежність несуттєва, то використовується модель стаціонарного каналу. Стаціонарний канал характеризується тим, що перехідні імовірності не залежать від часу. Нестаціонарний канал може бути поданий рядом стаціонарних каналів, які відповідають різним проміжкам часу.

Розрізняють також канали з пам'яттю *(channels with memory)* та канали без пам'яті *(memoryless channel)*. Для каналів з пам'яттю перехідні ймовірності в даному стані залежать від його попередніх станів.

Якщо перехідні імовірності постійні для всіх станів, то фактично канал має один стан і він називається стаціонарним каналом без пам'яті.

Під k-їчним каналом розуміють канал зв'язку, у якого кількість символів на вході і виході однакова і дорівнює k.

Розглянемо стаціонарний дискретний двійковий канал без пам'яті. Він однозначно визначається 4-ма умовними імовірностями $p_0(0)$, $p_1(1)$, $p_0(1)$, $p_1(0)$. Таку модель каналу зображають у вигляді графа (рис. 1.3).

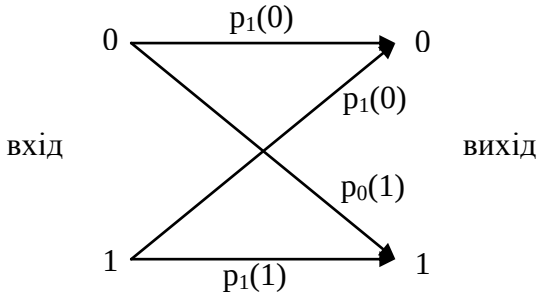


Рисунок 1.3 – Модель стаціонарного дискретного двійкового каналу без пам'яті

Якщо $p_1(0) = p_0(1) = q$, а $p_0(0) = p_1(1) = p$, то такий канал називається двійковим симетричним каналом *(binary symmetric channel)*. Символи на його вході правильно приймаються з імовірністю p і неправильно з імовірністю q=1-p. Це спрощує математичну модель каналу.

Ще однією моделлю дискретного каналу є канал зі стиранням. Для нього характерно, що алфавіт вихідних символів відрізняється від алфавіту вхідних символів. На вході, як і раніше, символи 0 та 1, а на виході каналу фіксується стан, при якому сигнал може бути віднесений як до одиниці, так і до нуля. Цей стан позначається символом стирання S. При декодуванні такі символи виправити легше, чим помилково визначені. Розрізняють моделі каналу зі стиранням при відсутності (рис. 1.4, а) і наявності (рис. 1.4, б) трансформації символів.

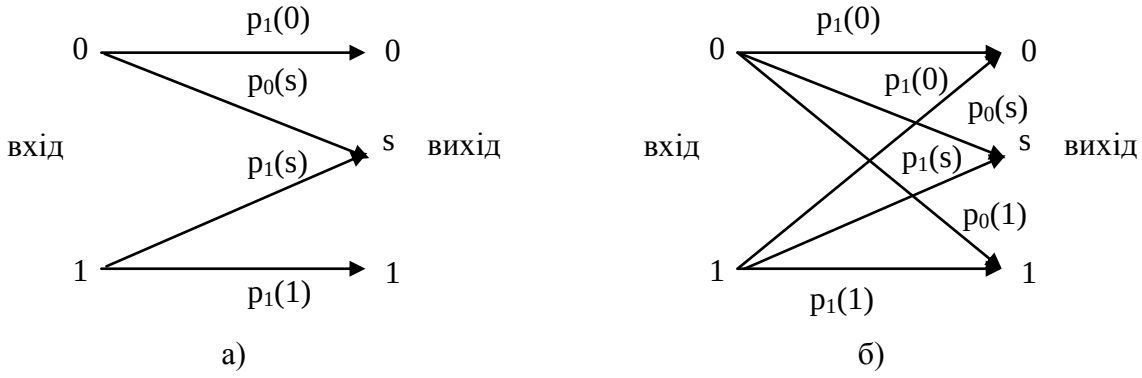


Рисунок 1.4 – Моделі каналів зі стиранням

1.12 Основні характеристики дискретного каналу

Швидкість передачі інформації по дискретному каналу. Для характеристики дискретного каналу використовують два поняття швидкості передачі [2].

1. Технічна швидкість передачі - швидкість маніпуляції *(manipulation speed)*:

$$V_\tau = \frac{1}{\tau_{ср}}, \qquad (1.39)$$

$\tau_{ср}$ - середнє значення тривалості символу. Одиницею вимірювання технічної швидкості є [бод] – швидкість при якій в секунду передається один символ.

2. Інформаційна швидкість або швидкість передачі інформації *(information rate)*. Визначається середньою кількістю інформації, яка передається по каналу в одиницю часу:

$$\bar{I}(V,U) = V_\tau \cdot I(V,U). \qquad (1.40)$$

де $I(V,U)$ - середня кількість інформації, яка переноситься одним символом.

Пропускна здатність дискретного каналу без завад. Пропускна здатність *(capacity)* дискретного каналу без завад визначається таким виразом [2]:

$$C_д = \max \bar{I}(V,U) = \max(V_\tau \cdot I(V,U)). \qquad (1.41)$$

Тобто, пропускна здатність дорівнює тій максимальній швидкості передачі інформації по даному каналу, якої можливо досягнути при найдосконаліших способах передачі і прийому, оскільки при відсутності завад має місце однозначна відповідність між множиною можливих V символів на виході і U на вході каналу, то:

$$I(U,V) = I(V,U) = H(U),$$

причому $H_{max}(U)=\log_2 m$, m – об’єм алфавіту символів. Тому для дискретного каналу без завад:

$$C_д = V_\tau \cdot \log_2 m. \qquad (1.42)$$

Тобто, для збільшення швидкості передачі інформації по дискретному каналу без завад і наближення її до пропускної здатності каналу послідовності букв повідомлення повинні так перетворюватись в кодері, щоб різні символи у вихідній послідовності появлялись по можливості рівноімовірно, а статистичні зв’язки між ними були відсутні.

Пропускна здатність дискретного каналу із завадами. При наявності завад відповідність між символами на вході і на виході каналу зв'язку перестає бути однозначною. В цьому випадку середня кількість інформації, що передається одним символом, визначається так:

$$I(V,\ U) = H(V)\ - H_U(V)\ = H(U) - H_V(U)\ = \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} p(v_j u_i)\log\frac{p(v_j u_i)}{p(u_i)p(v_j)}, \quad (1.43)$$

де $H_v(U)$ – умовна ентропія ансамблю U по відношенню до V;

$H_u(V)$ – умовна ентропія ансамблю V по відношенню до U;

m_1 – об'єм алфавіту вхідних повідомлень;

m_2 – об'єм алфавіту на виході каналу.

Таким чином швидкість передачі інформації по каналу із завадами:

$$\bar{I}(V,U) = V_\tau I(V,U) = V_\tau \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} p(v_j u_i)\log\frac{p(v_j u_i)}{p(u_i)p(v_j)}, \quad (1.44)$$

де V_τ - швидкість маніпуляції (технічна швидкість).

Оскільки, I(V,U) можна максимізувати, змінюючи статистичні властивості послідовностей символів на вході каналу за рахунок перетворення (кодера каналу), то отримане максимальне значення називають пропускною здатністю дискретного каналу зв'язку із завадами і позначається $C_д$

$$C_д = \max_{p\{U\}} V_\tau \cdot I(V,U), \quad (1.45)$$

де $p\{U\}$ - множина можливих розподілів імовірностей вхідних сигналів.

Пропускна здатність визначає найбільшу кількість інформації, яка може бути передана по каналу зв'язку із завадами в одиницю часу з як завгодно малою імовірністю помилки. Але граничні можливості каналу ніколи не викликаються. Ступінь завантаження каналу характеризує коефіцієнт його використання:

$$\lambda = \frac{\bar{I}(Z)}{C_д}, \quad (1.46)$$

де $\bar{I}(Z)$ - продуктивність джерела інформації.

Оскільки, як буде показано нижче, $0 \le \bar{I}(Z) \le C_д$, то $0 \le \alpha \le 1$ [2].

Контрольні питання і вправи

1. Дайте означення інформації.
2. Чим відрізняються такі поняття як інформація, повідомлення, сигнал?
3. Наведіть класифікацію повідомлень.
4. Охарактеризуйте кількісні оцінки інформації.
5. Поясніть який взаємозв'язок міри Шеннона і Хартлі.
6. Охарактеризуйте властивості ентропії.
7. Умовна ентропія і її властивості.
8. Кількість інформації як міра знятої невизначеності.
9. Модель системи передачі. Кодування джерела інформації та кодування каналу.
10. Моделі джерел дискретних повідомлень, властивості ергодичних послідовностей знаків.
11. Міра надлишковості. Продуктивність джерела дискретних повідомлень.
12. Моделі дискретних каналів.
13. Основні характеристики дискретного каналу - швидкість передачі інформації по дискретному каналу, пропускна здатність дискретного каналу без завад і з завадами.

Вправи

1. Визначити ентропію повідомлення, ймовірності появи символів якого такі: $p(z_1)= p(z_2)= p(z_3)= p(z_4)=0{,}01$, $p(z_5)=0{,}96$. Відповідь: 0,322 біт.
2. Визначити ентропію повідомлення "aaaabbcddd". Відповідь: 1,84 біт.
3. Визначити ентропію двійкового джерела інформації, якщо імовірність появи "0" – $p(0)=0{,}1$, а ймовірність появи "1" – $p(1)=0{,}9$. Відповідь: 0,47 біт.
4. Визначити ентропію джерела інформації, якщо імовірність появи на виході комбінації "00_2" – $p(00)=0{,}01$, "11_2" – $p(11)=0{,}81$, "01_2" – $p(01)=0{,}09$, "10_2" – $p(10)=0{,}09$. Пояснити результат та його взаємозв'язок з вправою 3. Відповідь: 0,94 біт.

5. Для двійкового джерела визначити ймовірності появи “0” та “1”, при яких ентропія максимальна. Відповідь: 0,5.
6. Визначити ентропію H(U), H(V), $H_V(U)$, H(UV), якщо задана матриця ймовірностей станів системи, яка об’єднує джерела U і V:

$$P(VU) = \begin{pmatrix} 0,3 & 0,1 & 0 \\ 0 & 0,2 & 0,2 \\ 0 & 0 & 0,2 \end{pmatrix}.$$

   Відповідь: H(U)=1,52 біт, H(V)=1,57 біт, $H_V(U)$=0,67 біт, H(UV)=2,24 біт.
7. Визначити чи є ергодичним стаціонарне дискретне джерело повідомлень, алфавіт якого складається з 4-х знаків z_1, z_2, z_3, z_4 Причому, безумовні імовірності вибору знаків такі:

$$p(z_1) = p(z_2) = p(z_3) = p(z_4) = \frac{1}{4}.$$

   А умовні ймовірності задані такою таблицею:

Таблиця 1.2 – Умовні імовірності вибору знаків

$z_q \backslash z_i$	z_1	z_2	z_3	z_4
z_1	1/2	1/4	1/4	0
z_2	1/4	1/4	1/2	0
z_3	1/4	1/2	1/4	0
z_4	0	0	0	1

   Відповідь: неергодичне.
8. Оцінити, яку частку загальної кількості можливих послідовностей необхідно врахувати в практичних розрахунках, якщо ергодичне джерело характеризується такими параметрами: n =8 (об’єм алфавіту джерела), H(Z) = 2 біти (ентропія джерела), N = 100 (кількість знаків в послідовності). Відповідь: $1/2^{100}$.
9. Визначити надмірність повідомлення “aaaabbcddd“ (вправа 2). Відповідь: 0,54.
10. Розробити програму для визначення частот символів у файлі. Мова програмування C++.

2 ЕФЕКТИВНЕ КОДУВАННЯ ІНФОРМАЦІЇ

2.1 Нерівномірні коди

Звичайно в комп'ютерних системах інформація подається рівномірними або блоковими кодами. Наприклад, в кодувальній таблиці KOI8-U кожному знаку алфавіту відповідає блок у 8 бітів. На відміну від рівномірних кодів нерівномірні коди характеризуються тим, що знаки повідомлення подаються кодами не обов'язково рівної довжини. Перевагою таких кодів є більша ефективність, оскільки при їх застосуванні для подання знаків повідомлення можна використовувати менше символів (бітів) [4]. Для цього необхідно мати відомості про статистику повідомлень, що посилаються. Якщо всі символи джерела інформації рівноімовірні, то виграш в порівнянні з нерівномірними незначний. Але якщо деякі знаки повідомлення більш імовірні в порівнянні з іншими, то це можна використати, подаючи знаки, які частіше зустрічаються, більш короткими кодовими комбінаціями. Тоді нерівномірні коди можуть бути суттєво більш ефективними в порівнянні з блоковими.

Однак, у випадку нерівномірних кодів, виникає фундаментальна проблема: як на приймальному кінці відрізнити окремі кодові слова? Наприклад, як в двійковій системі відрізнити, де закінчується одне кодове слово і починається наступне? Тому основною властивістю коду є однозначність декодування.

Розглянемо такий приклад. Нехай маємо код: $z_1 = 0$, $z_2=10$, $z_3 =110$, $z_4 =111$. Для декодування такого коду приймач використовує скінченний автомат, що еквівалентно дереву розв'язків на рис. 2.1.

Після прийому першої двійкової цифри автомат переходить з початкового стану в кінцевий стан z_1, якщо прийнято "0", або в другу точку розгалуження, якщо прийнято "1". Наступна прийнята двійкова цифра або переводить автомат в кінцевий стан z_2, якщо прийнято "0", або в третю точку розгалуження, якщо прийнято "1". Дальше, якщо прийнято "0", то автомат переходить в кінцевий стан z_3, а якщо прийнято "1", то автомат переходить в кінцевий стан z_4. В кожному кінцевому стані видається відповідний символ і автомат переходить в початковий стан, тобто кожний біт прийнятої послідовності переглядається лише один раз.

В цьому прикладі декодування є миттєвим [4], оскільки при отриманні всього кодового слова приймач зразу про це знає і йому нема необхідності досліджувати наступні біти, щоб прийняти рішення, який символ джерела прийнятий.

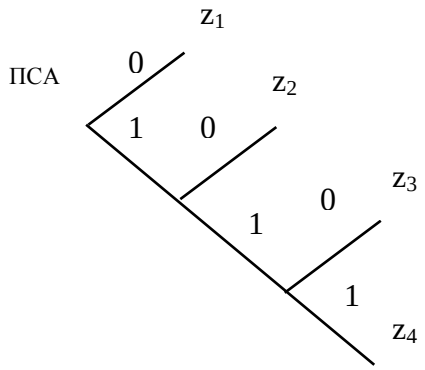


Рисунок 2.1 - Дерево декодування. ПСА –початковий стан автомата

Ніяке кодове слово цього коду не є префіксом (тобто, не збігається з початковою частиною) іншого кодового слова. Коди такого типу називаються миттєвими *(instantaneous code)* або префіксними *(prefix code)*. Крім того, розглянутий код називається “кодом з комою”, оскільки на кінці кожного кодового слова є двійкова цифра “0”.

Розглянемо ще один приклад. Нехай маємо код: $z_1 = 0$, $z_2=01$, $z_3 =011$, $z_4 =111$. Хоча цей код можна декодувати, але він не є миттєвим, оскільки деякі кодові слова є префіксами інших. Декодують цей код з кінця, тобто приймач повинен очікувати останній знак повідомлення, що вимагає в великого об’єму пам’яті.

Таким чином, для отримання миттєвого коду необхідно і достатньо щоб ніяке кодове слово, яке відповідає знаку z_i не було префіксом іншого кодового слова, що відповідає знаку z_j, а для немиттєвих кодів прийняття рішення деколи вимагає необмеженого об’єму пам’яті [4].

2.2 Нерівність Крафта

Нерівність Крафта дає умову існування миттєвих кодів, але не вказує як їх будувати [4].

Теорема. Необхідною і достатньою умовою існування миттєвого коду для джерела з алфавітом Z із k- символів z_i, де i = 1, 2, ... k, кодові слова якого мають довжини $l_1 \le l_2 \le l_3 \le \ldots \le l_k$, є виконання нерівності:

$$\sum_{i=1}^{k} \frac{1}{r^{l_i}} \le 1, \qquad (2.1)$$

де r – основа системи числення (коду).

Приклад 2.1. Нехай r=2 і задана довжина кодових слів (ДКС) - 1,3,3,3. Перевірити чи можна побудувати миттєвий код для таких довжин кодових слів.

Розв'язування. З нерівності Крафта (2.1) отримаємо:

$$\frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} = \frac{7}{8}.$$

Оскільки $\frac{7}{8}$<1, тобто нерівність Крафта виконується, то миттєвий код з такими довжинами існує.

Приклад2.2. Нехай r=2 і задана довжина кодових слів - 1,2,2,3. Перевірити чи можна побудувати миттєвий код для таких довжин кодових слів.

Розв'язування. З нерівності Крафта (2.1) отримаємо:

$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^3} = \frac{9}{8} > 1$$

Нерівність Крафта не виконується, тому миттєвий код з такими довжинами не існує.

2.3 Укорочені блокові коди

Нехай маємо 2^m кодових слів (або r^m в системі числення за основою r), кожне довжиною m-біт. Нехай число символів джерела не рівне точному степені основи. Розглянемо приклад.

Нехай кількість символів джерела дорівнює 5. Для подання цих символів потрібно три розряди, але три розряди дозволяють передавати 8 різних комбінацій. Необхідно відкинути 3 комбінації, але яких? Для вирішення цього питання побудуємо скінченний автомат для декодування цього коду (рис. 2.2).

Якщо відкинути 001, 011, 101, то можливо укоротити 3 гілки дерева, зберігши миттєве декодування (рис. 2.3, а).

Таким чином отримуємо S_1=00, S_2=01, S_3=10, S_4=110, S_5=111. Перевіряємо за нерівністю Крафта:

$$\frac{1}{2^2}+\frac{1}{2^2}+\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^3}=1.$$

Даний миттєвий код існує, оскільки виконується нерівність Крафта. Якщо вважати, що всі символи рівноімовірні, тоді середня довжина кодових комбінацій така:

$$l_{ср}=\sum_{i=1}^{5}p_i n_i = 0{,}2\cdot 2+0{,}2\cdot 2+0{,}2\cdot 3+0{,}2\cdot 3=2{,}4 \text{ біт}.$$

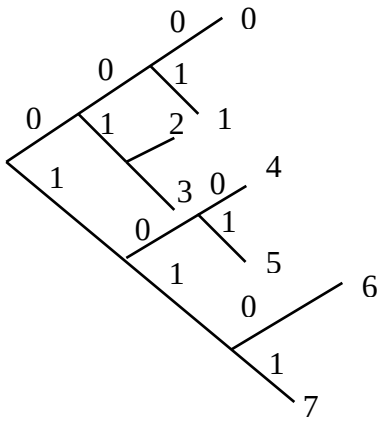


Рисунок 2.2 - Дерево декодування для трибітового рівномірного коду

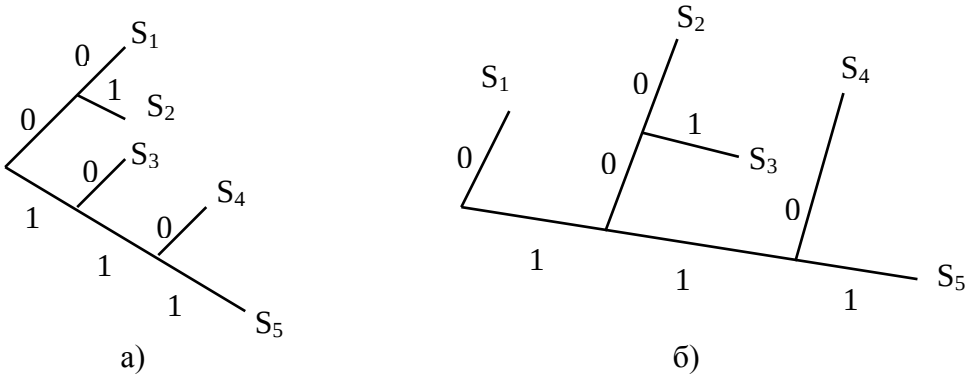


Рисунок 2.3 – Дерево декодування блокового укороченого коду

Розглянуті вище коди називають блоковими укороченими кодами [4].

Можна розглянути і інші варіанти, наприклад, відкинемо 001, 010, 011 (рис. 2.3,б). Отримаємо, S_1=0, S_2=100, S_3=101, S_4=110, S_5=111. Перевіряємо чи даний код є миттєвим:

$$\frac{1}{2}+\frac{1}{2^3}+\frac{1}{2^3}+\frac{1}{2^3}+\frac{1}{2^3}=1.$$

Отже, даний код миттєвий, тобто його можна однозначно декодувати. Якщо вважати, що всі символи рівноімовірні, тоді середня довжина кодових комбінацій така:

$$l_{ср}=\sum_{i=1}^{5}p_i n_i = 0{,}2\cdot 1+0{,}2\cdot 3+0{,}2\cdot 3+0{,}2\cdot 3=2{,}6 \text{ біт}.$$

Який із укорочених блокових кодів кращий, залежить від статистики джерела інформації, оскільки ми розглянули тільки випадок коли всі символи джерела інформації рівноімовірні.

2.4 Деякі коди змінної довжини

Гамма і дельта-коди Еліаса

Ці коди (табл. 2.1) генеруються так [10]:

Таблиця 2.1 – Гамма і дельта коди Еліаса

Діапазон	Гама-коди	Довжина коду, біт	Дельта-коди	Довжина коду, біт
1	1	1	1	1
2...3	01x	3	010x	4
4...7	001x	5	011xx	5
8...15	0001xxx	7	00100xxx	8
16...31	00001xxxx	9	00101xxxx	9
32...63	000001xxxxx	11	00110xxxxx	10
64...127	0000001xxxxxx	13	00111xxxxxx	11
128...255	00000001xxxxxxx	15	0001000xxxxxxx	14

Символами "x" тут позначено біти мантиси без старшої одиниці. Для діапазона $[2^K, 2^{K+1}-1]$ коди формуються таким чином:

- γ-код: 00...(K раз)...001x..(K раз)..x ; довжина: 2K+1 біт;
- δ-код: n...(2L+1 раз)...nx..(kраз)..x ; довжина: 2L+K+1 біт,

де $L= [1\log_2(K+1)]$ - ціла частина значення логарифма числа (K+1) за основою 2;

n - біти, які відносяться до запису експонент δ-коду, їх 2L+1.

Єдина відмінність між γ- і δ-кодами полягає у тому, що в γ-кодах експоненти записуються в унарному вигляді, а в δ-кодах до них ще раз застосовується γ-кодування.

Видно, що γ-коди перших 15 чисел коротші δ-кодів, а γ-коди перших 31 не довші δ-кодів. Тобто, чим нерівномірніший розподіл імовірностей, чим крутіше росте імовірність чисел при наближенні їх значення до 0, тим γ-коди кращі за δ-коди.

Коди Райса і Голомба

Коди Райса і Голомба (табл. 2.2) спочатку задаються з одним параметром і мають такий вигляд [10]:

Таблиця 2.2 – Коди Райса і Голомба

Код Голомба:	m=1	m =2	m=3	m=4	m=5	m=6	m=7	m=8
Код Райса:	k=0	k=1		k=2				k=3
n=1	0	00	00	000	000	000	000	0000
2	10	01	010	001	001	001	0010	0001
3	110	100	011	010	010	0100	0011	0010
4	1110	101	100	011	0110	0101	0100	0011
5	11110	1100	1010	1000	0111	0110	0101	0100
6	111110	1101	1011	1001	1000	0111	0110	0101
7	1111110	11100	1100	1010	1001	1000	0111	0110
8	…	11101	11010	1011	1010	1001	1000	0111
9	…	111100	11011	11000	10110	10100	10010	10000

Для m=2,3,4,8 алгоритм побудови коду можна зрозуміти за допомогою табл. 2.3.

Таблиця 2.3 – Побудова кодів Райса – Голомба при m=2,3,4,8

m=2	m=3	m=4	m=8
0x	00	0xx	0xxx
10x	01x	10xx	10xxx
110x	100	110xx	110xxx
1110x	101x	1110xx	1110xxx
11110x	1100	11110xx	11110xxx
	1101x		
	11100		
	11101x		

Видно, що коди Голомба при $m=2^S$ - це коди Райса з k = S, експоненти записуються в унарному вигляді, а під мантиси відведено S біт.

В табл. 2.4 наведено код Райса-Голомба при m=5,6,7.

Таблиця 2.4 – Побудова кодів Райса-Голомба при m=5,6,7

m=5	m=6	m=7
00x	00x	000
010	01xx	001 x
011x	100x	01 xx
100x	101xx	1000
1010	1100x	1001x
1011x	1101xx	101xx
1100x	11100x	11000
	11101xx	11001x
	111100x	1101xx
		111000

Якщо $m<2^S$, перші m кодів починаються з 0, друга група з m кодів починається з 10, третя — 110 і т. д. Діапазони довжиною m не вирівняні по границях, рівних 2^L, як в γ-кодах Еліаса. Експоненти обчислюються як e=(n-1)/m (цілочислове ділення) і записуються в унарній системі числення: e біт 1 і в кінці біт 0. Під мантиси r=n-em-1 відводиться або (S-1), або S біт.

Видно, що до експонент, як і в випадку з γ - кодами Еліаса, можна застосувати або γ(X) -, або Golomb(m) - кодування. Аналогічно і до експонент γ-коду - не тільки γ(X) -, але і Golomb(m) - кодування.

Омега-коди Еліаса і коди Івен-Роде

Ці коди (табл. 2.5) визначені так [10]:

Таблиця 2.5 – Коди Еліаса та Івен-Роде

Діапазон	ω-код	Біти	Івен-Роде-код	Біти
1	0	1	00	2
2...3	1x0	3	01x	3
4...7	10 1xx0	6	1xx0	4
8...15	11 1xxx0	7	100 1xxx0	8
16...31	10 100 1xxxx0	11	101 1xxxx0	9
32.. .63	10 101 1xxxxx0	12	ПО 1xxxxx0	10
64. ..127	10 110 1xxxxxx0	13	1 1 1 1xxxxxx 0	11
128...255	10111 1xxxxxxx 0	14	100 1000 1xxxxxxx0	16

І ті, і інші складаються з послідовності груп, довжиною L_1, L_2, L_3,…, L_m і починаються з біта 1. Кінець послідовності задається бітом 0. Довжина

кожної наступної (n+1)-ої групи задається значенням бітів попередньої n-ої групи. Значення бітів останньої групи є результатом всього коду, тобто, всієї послідовності груп. Інакше кажучи, всі перші m-1 груп служать тільки для визначення довжини останньої групи.

У ω-кодах Еліаса довжина першої групи - 2 біта і далі довжина наступної групи дорівнює значенню попередньої плюс один. Перше значення задано окремо.

В Івен-Роде кодах довжина першої групи - 3 біта і далі довжина кожної наступної групи дорівнює значенню попередньої. Перші 3 значення задані особливим способом.

При кодуванні формується спочатку остання група, потім передостання і т. д., поки процес не буде завершений.

При декодуванні, навпаки, спочатку зчитується перша група, за значенням її бітів визначається довжина наступної групи (якщо перша группа починається з одиниці) або кінцеве значення коду (якщо група починається з нуля).

Старт-крок-стоп (start-step-stop) коди

Ці коди ще називаються кодами цілих змінної довжини *(Variable Length Integers (VLI) codes)* [10].

Старт-крок-стоп-коди задаються трьома параметрами: (i,j,k).

Експонента може займати 1, 2, 3,..., m-1, m біт, а мантиса – i, i+j, i+2j, i+3j,…, k біт. Якщо (i, j, k)=(3, 2, 11), тоді коди мають вигляд, наведений в табл. 2.6.

Таблиця 2.6– Старт-крок-стоп коди

Діапазон	(3, 2, 11)-код	Довжина коду, біт
1...8	0xxx	4
9...40	10xxxxx	7
41...168	110xxxxxxx	10
169...680	1110xxxxxxxxx	13
681. ..2728	1111xxxxxxxxxxx	15

Таким чином, цю відповідність можливо використовувати для чисел з діапазону [1,2728]. Експонента записується в унарній системі числення: кінець поля експоненти вказується за допомогою нуля. Для п’ятої групи, що має максимальну довжину мантиси - 11 біт, 0 що розділяє не потрібен через те, що незалежно від його наявності декодування однозначне.

Якщо максимальне значення не задано, тоді третій параметр не задається. Такі коди називаються старт-крок-кодами (start-step-codes).

Коди Фібоначі

Початкове число N розкладається в суму чисел Фібоначі f_i (f_1=1, f_2=2, …, $f_i=f_{i-1}+f_{i-2}$). Відомо, що будь-яке число однозначно подається у вигляді суми чисел Фібоначі. Тому можна побудувати код числа як послідовність бітів, кожний з яких указує на факт наявності в *N* певного числа Фібоначі. Відзначимо також, що якщо в N є f_i, то в ньому не може бути f_{i+1}. Тому якщо одиничне значення біта указує на використання якогось числа f_i, то ми можемо позначати кінець запису поточного коду і початок наступного послідовністю з двох одиниць(табл.. 2.7):

Таблиця 2.7 – Коди Фібоначчі

f_i.:	1	2	3	5	8	13	21	34	55
n=1	1	(1)							
2	0	1	(1)						
3	0	0	1	(1)					
4	1	0	1	(1)					
5	0	0	0	1	(1)				
6	1	0	0	1	(1)				
7	0	1	0	1	(1)				
8	0	0	0	0	1	(1)			
…	…	…	…	…	…	…			
12	1	0	1	0	1	(1)			
13	0	0	0	0	0	1	(1)		
…	…	…	…	…	…	…	…		
20	0	1	0	1	0	1	(1)		
21	0	0	0	0	0	0	1	(1)	
…	…	…	…	…	…	…	…	…	
27	1	0	0	1	0	0	1	(1)	

Як і у випадку з унарним записом, немає чіткого розділення на мантиси і експоненти. Можна вважати, що при записі в унарному вигляді всі біти, окрім останнього, експоненти. А в кодах Фібоначчі навпаки: всі біти, окрім двох останніх, мантиси.

Розглянемо твердження докладніше. Якщо в потоці γ-кодів або кодів Райса буде спотворено один біт, довжина і вміст решти потоку не

зміниться, якщо цей біт мантиса (і "зіпсованим" стане тільки один код). Але якщо "зламаний" біт експонента, то буде неправильно декодована значна частина потоку.

Якщо в потоці унарних кодів змінити один біт, кодів стане або на один більше, якщо цей біт експонента:

...00000001000000001... - було

... 00100001000000001... – стало,

або на один код менше, якщо цей біт мантиса: ...00000000000000001...

З іншого боку, для потоку кодів Фібоначчі кодів стане на один менше, якщо біт, що "зламався", експонента, тобто один з двох одиничних бітів, що позначають кінець коду, або на один більше, якщо "збійний" біт став таким, як біт експоненти, і хоч би один з двох сусідніх із "збійним" бітів одиничний. У решті випадків "зламається" тільки один код (в деяких випадках може "зламатися" пара сусідніх кодів). Але весь потік, подібно до γ- і Голомб-кодів, ніколи [11].

2.5 Основна теорема Шеннона для кодування каналу без завад

Ефективне кодування повідомлень для передачі їх по дискретному каналу без завад ґрунтується на теоремі Шеннона, яку можна сформулювати так [1-2]:

- повідомлення джерела з ентропією H(z) завжди можна закодувати послідовностями символів з об’ємом алфавіту m так, що середнє число символів на знак повідомлення $l_{ср}$ буде як завгодно близьким до величини $\frac{H(z)}{\log m}$, але не менше за неї. При m=2, $l_{ср} \geq H(z)$.

Теорема не вказує конкретного способу кодування, але з неї видно, що кожний символ кодової комбінації повинен нести максимальну інформацію, тобто кожний символ повинен приймати значення або 0, або 1 по можливості з рівними імовірностями і кожний вибір повинен бути незалежним від значення попередніх символів. Для випадку відсутності взаємозв’язків між знаками конструктивні методи побудови ефективних кодів розроблені американськими вченими Шенноном і Фано. Оскільки їх методики подібні, то відповідний метод отримав назву Шеннона-Фано [2].

2.6 Ефективне кодування некорельованої послідовності знаків методом Шеннона-Фано

Згідно з методом Шеннона-Фано код будують таким чином [2].

1. Знаки алфавіту повідомлення вписують в таблицю в порядку убування їх імовірностей
2. Потім їх розділяють на 2 групи так, щоб суми імовірностей в обох групах були по можливості однакові.
3. Всім знакам однієї групи приписують як перший символ 0(1), а знакам другої групи приписують символ 1(0).
4. Кожну з отриманих груп знову аналогічно розбивають на 2 підгрупи.
5. Процес повторюється доти, поки в кожній підгрупі не залишиться по одному знаку.

Приклад 2.3. Виконати ефективне кодування ансамблю із 8 знаків, якщо відомі їх імовірності.

Розв'язування. Використовуючи методику Шеннона-Фано, отримаємо сукупність кодових комбінацій, наведених в табл. 2.8.

Таблиця 2.8 – Кодування Шеннона-Фано

Знак	p	Кодові комбінації	Ступінь
z_1	1/2	1	I
z_2	1/4	01	II
z_3	1/8	001	III
z_4	1/16	0001	IV
z_5	1/32	00001	V
z_6	1/64	000001	VI
z_7	1/128	0000001	VII
z_8	1/128	0000000	VII

Обчислимо ентропію повідомлення:

$$H(z) = -\sum_{i=1}^{8} p(z_i) \log p(z_i) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \ldots - \frac{1}{128}\log\frac{1}{128} = \frac{127}{64} \quad \text{біт.}$$

А середня кількість символів на знак після кодування визначається так:

$$l_{ср} = \sum_{i=1}^{8} p(z_i) n(z_i),$$

де $n(z_i)$ – кількість символів (розрядів) в кодовій комбінації, що відповідає знаку z_i.

$$l_{ср} = 1/2 \cdot 1 + 1/4 \cdot 2 + \ldots + 1/128 \cdot 7 = 127/64 \text{ біт.}$$

Оскільки імовірності знаків є цілочислові від'ємні значення двійки, то надлишковість при кодуванні вилучається повністю, що і підтверджує рівність ентропії середній кількості символів на знак після кодування.

Приклад 2.4. Розглянемо процедуру ефективного кодування повідомлень утворених за допомогою алфавіту, що складається з двох знаків z_1 і z_2 з імовірностями появи $p(z_1)=0,9$, $p(z_2)=0,1$.

Розв'язування. Визначимо ентропію джерела інформації:

$$H(Z) = -0,9 \log_2 0,9 - \log_2 0,1 = 0,47 \text{ біт.}$$

Однак на передачу однієї букви потрібен 1 біт. Щоб видалити цю надмірність перейдемо до кодування блоками з урахуванням того, що знаки статистично незалежні:

$$p(z_m z_n) = p(z_m) \quad p(z_n).$$

Таблиця 2.9 – Кодування Шеннона-Фано блоками по 2 символи

Блоки	p	Кодові комбінації	Ступінь розбиття
$z_1 z_1$	0,81	1	I
$z_1 z_2$	0,09	01	II
$z_2 z_1$	0,09	001	III
$z_2 z_2$	0,01	000	III

Розрахуємо середню кількість біт на блок повідомлення:

$$l_{срб} = \sum_{i=1}^{4} p(z_i z_j) n(z_i z_j) == 0,81 \cdot 1 + 0,09 \cdot 2 + 0,09 \cdot 3 + 0,01 \cdot 3 = 1,29 \text{ біт.}$$

Оскільки символів в блоці 2, то середня кількість біт на символ повідомлення складе:

$$l_{ср} = l_{срб}/2 = 0,645 \text{ біт.}$$

Кодування блоків, що містить 3 знаки дасть ще кращий результат.

Таблиця 2.10 – Кодування Шеннона-Фано блоками по 3 символи

Блоки	p	Кодові комбінації	Ступінь розбиття
$z_1 z_1 z_1$	0,729	1	I
$z_2 z_1 z_1$	0,081	011	III
$z_1 z_2 z_1$	0,081	010	III
$z_1 z_1 z_2$	0,081	001	III
$z_2 z_2 z_1$	0,009	00011	V
$z_2 z_1 z_2$	0,009	00010	V
$z_1 z_2 z_2$	0,009	00001	V
$z_2 z_2 z_2$	0,001	00000	V

Розглянута методика Шеннона-Фано не завжди приводить до однозначної побудови коду. При розбитті на підгрупи можна зробити

більшою за імовірностями як одну, так і іншу підгрупу. Цих недоліків не має методика Хаффмана.

2.7 Типова класифікація методів оптимального кодування

Теорія економного або оптимального кодування *(optimal coding)* об'єднує в собі декілька різних напрямів. В рамках даної теорії методи прийнято розділяти на методи економного кодування інформації без втрат (*lossless)* і методи економного кодування інформації з втратами (*lossy)* [9, 12-14]. Як випливає з назв, обробка інформації методами першої групи не веде до інформаційних втрат, тоді як використання методів другої групи пов'язане з такими втратами. Розглянемо більш детально методи першої групи.

Серед алгоритмів ущільнення без втрат дві схеми ущільнення - кодування Хаффмана (Huffman) і LZW-кодування (за початковими літерами прізвищ Лемпел (Lempel) і Зів (Ziv) (його авторів) і Уелч (Welch), який його суттєво модифікував), формують основу для багатьох систем ущільнення [12]. Ці схеми подають два різних підходи до ущільнення даних. Окремий клас утворюють методи, які відомі як арифметичне кодування. Таким чином алгоритми ущільнення даних без втрат можна розділити на такі групи:

- статистичні методи ущільнення;
- словникові (евристичні) методи ущільнення;
- арифметичне кодування.

Статистичні алгоритми (Шеннона-Фано, Хаффмана, кодування за ступенем новизни, імовірнісне ущільнення та ін.) потребують знання імовірностей появи символів в повідомленні, оцінкою якої є частота появи символів у вхідних даних [12]. Як правило, ці імовірності невідомі. З урахуванням цього статистичні алгоритми можна поділити на три класи.

1. Неадаптивні - використовують фіксовані, завчасно задані імовірності символів. Таблиця імовірностей символів не передається разом з файлом, оскільки вона відома завчасно. Недолік: невеликий перелік файлів, для яких досягається прийнятний коефіцієнт стиснення.

2. Напівадаптивні - для кожного файла будується таблиця частот символів і за її допомогою ущільнюють файл. Разом з ущільненим файлом передається таблиця символів. Такі алгоритми досить непогано ущільнюють більшість файлів, але необхідна додаткова передача таблиці частот символів, а також два проходи початкового файлу для виконання кодування.
3. Адаптивні - починають працювати з фіксованою початковою таблицею частот символів (зазвичай всі символи спочатку рівноімовірні) і в процесі роботи ця таблиця змінюється в залежності від символів, що зустрічаються у файлі. Переваги: однопрохідність алгоритму, не потребує передачі таблиці частот символів, достатньо ефективно стискає широкий клас файлів.

Евристичні (словникові) алгоритми ущільнення (типу LZ77, LZ78), як правило, шукають в файлі рядки, що повторюються, і будують словник фраз, що вже зустрічались. Зазвичай такі алгоритми мають цілий ряд специфічних параметрів (розмір буфера, максимальна довжина фрази і т.п.), підбір яких залежить від досвіду автора роботи, і ці параметри добираються таким чином, щоб досягти оптимального співвідношення часу роботи алгоритму, коефіцієнта стиснення та переліку файлів, що добре стискаються [12].

Арифметичне кодування, подібно до статистичних методів, використовує як основу технології ущільнення, імовірність появи символу у файлі, однак сам процес арифметичного кодування має принципові відмінності. У результаті арифметичного кодування символьна послідовність (рядок) замінюється дійсним числом більше нуля і менше одиниці [12].

2.8 Статистичні алгоритми ущільнення даних без втрат

Алгоритм RLE

Даний алгоритм надзвичайно простий в реалізації. RLE *(Run Length Encoding)*, або групове кодування, один з найбільш давніх алгоритмів кодування графіки. Його суть полягає в заміні символів, що повторюються, на один цей символ та лічильник повторень. Проблема є в тому, щоб архіватор при відновленні міг відрізнити в результувальному потоці таку

кодовану серію від інших символів. Рішення очевидне - помістити у всі ланцюжки деякі заголовки (наприклад, використати перший біт як ознаку кодування серії). Метод достатньо ефективний для графічних зображень в форматі «байт *(byte)* на піксел» (наприклад, формат PCX використовує кодування RLE) .

Проте недоліки алгоритму очевидні - це, зокрема, мала пристосованість до багатьох типів файлів, що часто зустрічаються, наприклад, до текстових. Тому його можна ефективно використовувати тільки в поєднанні зі вторинним кодуванням. Цей підхід знайшов своє відображення в алгоритмі кодування факсів: спочатку зображення розбивається на чорні та білі точки, які перетворюються алгоритмом RLE в потік довжин серій, а потім ці довжини серій кодуються методом Хаффмана зі спеціально підібраним (експериментально) деревом [12].

Кодування Хаффмана

Хаффман запропонував будувати кодове дерево не зверху вниз, як це неявно робиться в алгоритмі Шеннона-Фано, - від кореневого вузла до листкових вузлів, а від низу до верху - від листкових вузлів до кореневого вузла. Такий підхід виявився найбільш ефективним і набув вельми широкого поширення.

На початковому етапі роботи алгоритму кожному символу інформаційного алфавіту ставиться у відповідність вага, рівна імовірності (частоті) появи даного символу в повідомленні. Символи поміщаються в список, який сортується за спаданням ваг. На кожному кроці (ітерації) два останні елементи списку об'єднуються в новий елемент, який потім поміщається в список замість двох об'єднуваних елементів. Новому елементу списку ставиться у відповідність вага, рівна сумі ваг тих елементів, що заміщаються. Кожна ітерація закінчується впорядковуванням отриманого нового списку, який завжди містить на один елемент менше, ніж старий список.

Паралельно з роботою вказаної процедури здійснюється послідовна побудова кодового дерева. На кожному кроці алгоритму будь-якому елементу списку відповідає кореневий вузол бінарного дерева, що складається з вершин, відповідних елементам, об'єднанням яких був отриманий даний елемент. При об'єднанні двох елементів списку відбувається об'єднання відповідних дерев в одне нове бінарне дерево, в

якому кореневий вузол відповідає новому елементу, що поміщається в список, а елементам списку, що заміщаються, відповідають дочірні вузли цього кореневого вузла. Алгоритм завершує роботу, коли в списку залишається один елемент, відповідний кореневому вузлу побудованого бінарного дерева. Це дерево називається деревом Хаффмана.

Система префіксних кодів може бути отримана шляхом присвоєння конкретних двійкових значень ребрам цього дерева. Приклад побудови дерева Хаффмана наведений на рис. 2.4.

Система кодів, отримана в результаті побудови дерева Хаффмана, є оптимальною системою префіксних кодів [13-14]. Звідси, проте, у жодному випадку не слідує висновок про те, що префіксні системи, отримані з використанням інших алгоритмів, завжди менш ефективні.

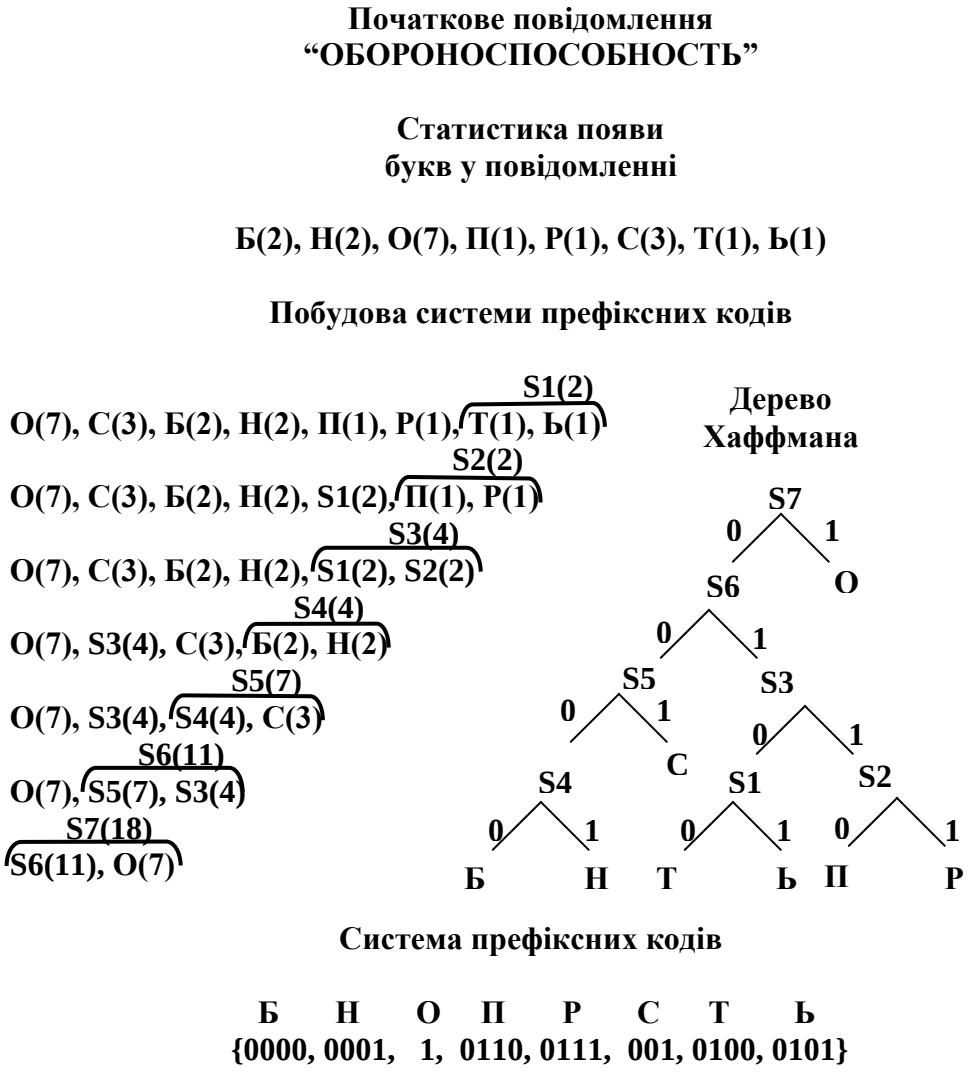


Рисунок 2.4 – Ілюстрація роботи алгоритму Хаффмана

Недоліком такого кодування є той факт, що разом із закодованим повідомленням необхідно передавати також і побудовану таблицю кодів (дерево), що знижує величину ущільнення. Однак існує динамічний алгоритм побудови дерева Хаффмана, у якому кодова таблиця обновляється самим кодувальником і синхронно й аналогічно декодувальником у процесі роботи після одержання кожного чергового символу. Одержувані при цьому коди виявляються квазіоптимальними, тому текст ущільнюється трохи гірше. Більш того, зростають складність алгоритму і час роботи програми (хоча вона і стає однопрохідною) [14].

Кодування за ступенем новизни

Одним із варіантів адаптивного алгоритму Хаффмена є алгоритм кодування за ступенем новизни, відомий також як MTF *(Move To Front - рухай нагору)*, або кодування за допомогою купки книжок [12,15]. Ідея методу така: нехай алфавіт джерела складається з N символів з номерами 1,2,...,N. Кодер зберігає список символів, які являють собою деяку перестановку алфавіту. Коли на вхід поступає символ C, який має в списку номер “i”, кодер передає код номеру “i”. Потім кодер переставляє символ C на початок списку, збільшуючи на одиницю номери всіх символів, які стоять перед C. Таким чином, більш "популярні" символи будуть тяжіти до початку списку та будуть мати більш короткі коди.

Як код для кодування за ступенем новизни можна використовувати монотонний код - універсальний код джерела, для якого відомо лише впорядкованість імовірностей символів. Монотонний код побудований так, що більш близькі до початку списку позиції, отримують більш короткі коди. В результаті літери, які часто з'являються та тяжіють до початку списку, будуть мати короткі кодові позначення. Розглянемо ущільнення інформації методом кодування за ступенем новизни на прикладі.

Нехай необхідно передати слово w=221312233 в алфавіті A={1,2,3}. Монотонним кодом позиції 1 є 0, позиції 2 - 10, позиції 3 - 11. Купка книжок при послідовній появі літер слова w змінюється таким чином:

(1,2,3) -- (2,1,3) -- (2,1,3) -- (1,2,3) -- (3,1,2) і т.д.

Кодом слова буде

100101110110…

Кодування за ступенем новизни ненабагато гірше найкращого кодування методом Хаффмана. Даний метод уступає кодуванню Хаффмана при ущільненні текстових файлів, однак, для файлів з малим початковим алфавітом, можна досягти гарних результатів. Кодування за ступенем новизни не потребує попереднього перегляду початкового масиву. Цей метод легко пристосовується до можливих коливань частот, тому він названий ще – локально-адаптивним. Крім-того, він ідеально підходить до сумісного використання разом з BWT-перетворенням, яке розміщує однакові символи блока, що перетворюється, один за одним.

Імовірнісне кодування

Надзвичайно цікавий і найшвидший з відомих методів ущільнення інформації, що дає до того ж непогані результати. Імовірнісне ущільнення багато в чому нагадує гадання на кавовій гущі або передбачення погоди, тільки більш ефективно [16].

Працює алгоритм в такий спосіб: є досить велика таблиця передбачень, яка досить швидко оновлюється, в якій для кожної можливої пари послідовних вхідних символів вказується передбачений наступний (третій) символ. Якщо символ передбачений правильно - генерується код у вигляді однобітного префікса, рівного 1. Якщо ж символ не вгадано - видається код у вигляді префікса, рівного 0 і не вгаданого символу. При цьому не вгаданий символ замінює в таблиці символ, що був там до цього, забезпечуючи постійне відновлення статистичної інформації [1].

Алгоритм є адаптивним і тому він однопрохідний і не вимагає збереження таблиці разом із закодованим текстом.

Декомпресор працює за аналогічною програмою, підтримуючи й оновлюючи таблицю синхронно з компресором. Якщо надійшов префікс 1, чергова вихідна буква береться з таблиці за індексом, отриманим з двох попередніх букв, інакше просто копіюється код із вхідного потоку у вихідний. Для невеликого підвищення ступеня ущільнення рекомендується ініціалізувати таблицю перед початком роботи будь-якими сполученнями, які часто зустрічаються.

Схожий алгоритм використовується в архіваторі PKZIP Ver.1.5 під ім'ям Reducing. Він майже за всіма параметрами гірший, ніж імовірнісний (двопрохідний, вимагає передачі таблиці разом з текстом), проте заслуговує на згадування. Таблиця програми Reducing містить масив

символів V[256] [32], тобто для кожної вхідної букви на першому проході знаходяться не більше, ніж 32 (але може бути і менше) наступних літер, які часто зустрічаються**.** На другому проході подібно до імовірнісного, якщо черговий символ b, що поступає за символом a знаходиться серед 32 очікуваних, генерується код <prefix, position>, prefix=1, position дорівнює положенню символу b у масиві V[*a*][ ] з 32 букв. Інакше видається префікс 0 і сам символ b. Алгоритм статичний, тому таблиця не оновлюється в процесі роботи.

2.9 Словникові методи оптимального кодування

Словникові алгоритми призначені для усунення надлишку ланцюжків подій. Суть алгоритму полягає в тому, що ланцюжки подій поміщаються в словник. Якщо надалі у вхідному потоці зустрічається ланцюжок подій, який присутній в словнику, то замість цього ланцюжка у вихідний потік поміщається посилання на елемент словника. Ідея словникових алгоритмів була запропонована Я. Зівом і А. Лемпелем в 1977 р. [13]. Ідея була втілена в алгоритмі LZ77, що став основою цілого сімейства алгоритмів. Альтернативний підхід, запропонований ними ж в 1978 р., породив сімейство алгоритмів LZ78.

Словникові алгоритми мають не найвищі коефіцієнти ущільнення, але допускають дуже швидку реалізацію, особливо при декодуванні даних, що і зумовлює їх широке використання.

Під словником в алгоритмах сімейства LZ77 розуміється деяка кількість попередніх подій. Як посилання на елементи словника використовуються відстань до ланцюжка подій, що зустрічалися раніше, і довжина цього ланцюжка [2].

Природно, що не для всіх ланцюжків подій вхідного потоку можна знайти відповідні ланцюжки в словнику. Щоб забезпечити відмінність між ланцюжками подій і літеральними подіями (подіями, що не увійшли до ланцюжків із словника), алгоритм LZ77 просто чергує посилання на елементи словника з літеральними подіями.

У алгоритмі LZSS посилання на елементи словника і літеральні події розрізняються прапорцем-бітом.

Алгоритм LZH аналогічний LZSS, але використовує для покажчиків кодування Хаффмана.

В наш час алгоритми сімейства LZ77 широко використовуються в програмах універсальних архіваторів. Це, як правило, модифікації LZSS з використанням кодів Хаффмана (LHARC) Шеннона-Фано (PKZIP) або арифметичного кодування (HA).

У алгоритмах сімейства LZ78 ланцюжки в словнику збільшуються на один символ кожного разу, коли ланцюжок із словника зустрічається у вхідному потоці. У алгоритмі LZ78, так само, як і в LZ77, літеральні події і елементи словника чергувалися.

У 1984 р. Т. Велч запропонував алгоритм LZW у якому всі літеральні події є елементами словника [12]. Таким чином, у вихідний потік поміщаються тільки посилання на елементи словника, що спрощує алгоритм. Велч показав, що алгоритм може бути легко реалізований апаратно. Програмна реалізація, як правило, виконується на основі вектора двійкових дерев. Корінням дерев служать елементи вхідного алфавіту.

LZC – модифікація алгоритму LZW направлена на підвищення коефіцієнта стиснення. По-перше, в ньому використовується змінна довжина кодів. Наприклад, при ущільненні зображень з 4 бітами на піксель використання 5-бітових кодів при порожньому словнику дає значну економію в порівнянні з 12-бітовими. У міру заповнення словника довжина кодів збільшується. По-друге, відстежується ступінь ущільнення алгоритму. Якщо вона падає нижче певної межі, словник повністю очищається. Цим досягається швидка адаптація алгоритму до різкої зміни характеру даних.

LZC використовується в утиліті COMPRESS операційної системи UNIX і графічному форматі GIF.

Алгоритм LZFG в деякому розумінні є гібридом LZ77 і LZ78. Як і LZ77 він допускає кодування ланцюжків довільної довжини (у алгоритмах LZ78 довжина ланцюжків, як правило, обмежена кількістю попередніх поява ланцюжка у вхідному потоці). З іншого боку, адресація елементів словника аналогічна техніці LZ78 з тією різницею, що замість двійкового дерева використовується так зване дерево Patricia. Його відмінність полягає в тому, що якщо дерево не містить розгалужень, то послідовність вершин об'єднується в одну.

Алгоритм RFGD є модифікацією LZFG з вищим ступенем стиснення [1]. Він поміщає у вихідний потік дані чотирьох типів: термінальні вершини дерева, нетермінальні вершини дерева, літерали, що зустрічалися

раніше у вхідному потоці і літерали, що раніше не зустрічалися. Кожний з них кодується за власною схемою. Для визначення того, якого саме типу дані поміщаються у вихідний потік, безпосередньо перед ними у вихідний потік поміщається спеціальний префікс. Алгоритм реалізований апаратно.

Алгоритм LZW

Алгоритм дуже простий. Якщо коротко, то LZW-стиснення заміняє рядки символів деякими кодами. Це робиться без будь-якого аналізу вхідного тексту. Замість цього, при добавленні кожного нового рядка проглядається таблиця рядків. Стиснення відбувається, коли код заміняє рядок символів. Коди, генеровані LZW-алгоритмом, можуть бути будь-якої довжини, але вони повинні містити більше біт, ніж одиничний символ. Перші 256 кодів (коли використовуються 8-бітові символи) за замовчуванням відповідають стандартному набору символів. Решта кодів відповідають рядкам, що обробляються алгоритмом.

При стисненні та відновленні LZW маніпулює трьома об`єктами: потоком символів, потоком кодів і таблицею рядків. При стисненні потік символів є вхідним і потік кодів - вихідним. При відновленні вхідним є потік кодів, а потік символів - вихідним. Таблиця рядків породжується і при стисненні і при відновленні, однак вона ніколи не передається від стиснення до відновлення і навпаки.

Оскільки ми не знаємо наперед об`єм файла, то важко визначити, яка розрядність кодів буде оптимальною. Тому доцільно використовувати коди різної довжини.

Ущільнення інформації. Наведемо алгоритм в найпростішій формі. Кожний раз, коли генерується новий код, новий рядок добавляється в таблицю рядків. LZW постійно перевіряє, чи даний рядок вже відомий, і, якщо так, виводить код без генерування нового.

Процедура LZW-стиснення наведена на рис. 2.5. Простий рядок, використаний для демонстрації алгоритму, наведений на рис. 2.6. Вхідний рядок є коротким списком англійських слів, розділених символом «/». Як ви можете помітити, аналізуючи алгоритм, його робота починається з того, що на першому кроці циклу він виконує перевірку на наявність рядка «/W» в таблиці.

Процедура LZW-стиснення:
РЯДОК = черговий символ з вхідного потоку
WHILE вхідний потік не пустий DO
СИМВОЛ = черговий символ з вхідного потоку
IF РЯДОК + СИМВОЛ в таблиці рядків THEN
РЯДОК = РЯДОК + СИМВОЛ
ELSE
вивести в вихідний потік код для РЯДОК
добавити в таблицю рядків РЯДОК + СИМВОЛ
РЯДОК = СИМВОЛ
END of IF
END of WHILE
вивести в вихідний потік код для РЯДОК

Рисунок 2.5 - Алгоритм ущільнення методом LZW

Коли він не знаходить цей рядок, то генерує код для «/» і добавляє в таблицю рядок «/W». Оскільки 256 символів вже визначені для кодів 0 - 255, то першому визначеному рядку може бути поставлено у відповідність код 256. Після цього система читає наступну букву «E», добавляє другий підрядок «WE» в таблицю рядків і виводить код для букви «W». Коли знову буде прочитаний рядок, що складається з символів «/» і «W», то йому буде поставлено у відповідність рядок з таблиці рядків. В цьому випадку система виводить код 256 і добавляє трисимвольний рядок в таблицю (/WE). Цей процес продовжується до тих пір, поки не закінчиться вхідний потік і всі коди не будуть виведені.

Вихідний потік для заданого рядка (/WED/WE/WEE/WEB/WET) показаний на рис. 2.6 так, як і отримана в результаті таблиця рядків. Ця таблиця швидко заповнюється, оскільки новий рядок добавляється в таблицю кожний раз, коли генерується код. В цьому прикладі було виведено п`ять закодованих підрядків і сім символів. Звичайно, цей приклад було вибрано тільки для демонстрації. В дійсності стиснення не починається до тих пір, поки не буде збудована досить велика таблиця, зазвичай після читання порядку 100 вхідних байтів.

Вхідний рядок: /WED/WE/ WEE/ WEB/WET

Вхід (символи)	Вихід (коди)	Нові коди і відповідні рядки
/W	/	256 = /W
E	W	257 = WE
D	E	258 = ED
/	D	259 = D/
WE	256	260 = /WE
/	E	261 = E/
WEE	260	262 = /WEE
/W	261	263 = E/W
EB	257	264 = WEB
/	B	265 = B/
WET	260	266 = /WET
<EOF>	T	

Рисунок 2.6 - Процес ущільнення

Відновлення інформації. Алгоритму стиснення відповідає свій алгоритм відновлення інформації. Він отримує вихідний потік кодів від алгоритму стиснення і використовує його для точного відновлення вхідного потоку. Однією з причин ефективності LZW-алгоритму є те, що він не потребує зберігання таблиці рядків, яка отримана при стисненні. Таблиця може бути точно відновлена на основі вихідного потоку алгоритму стиснення. Це можливо тому, що алгоритм стиснення виводить РЯДКОВУ та СИМВОЛЬНУ компоненти коду раніше, ніж він помістить цей код у вихідний потік. Це означає, що стиснені дані не обтяжені необхідністю тягнути за собою велику таблицю рядків.

Алгоритм декодування даних, стиснутих методом LZW, наведений на рис. 2.7. Відповідно до алгоритму стиснення, він добавляє новий рядок в таблицю рядків кожний раз, коли читає з вхідного потоку новий код. Все, що йому необхідно зробити додатково - це перевести кожний вхідний код в рядок і переслати його в вихідний потік. Важливо відзначити, що побудова таблиці рядків алгоритмом декодування закінчується тоді, коли побудована таблиця рядків алгоритмом стиснення.

Процедура LZW - декодування:
читати СТАРИЙ_КОД
вивести СТАРИЙ_КОД
WHILE вхідний потік не пустий DO
читати НОВИЙ_КОД
РЯДОК = перевести НОВИЙ_КОД
вивести РЯДОК
СИМВОЛ = перший символ РЯДКА
добавити в таблицю рядків СТАРИЙ_КОД + СИМВОЛ
СТАРИЙ_КОД = НОВИЙ_КОД
END of WHILE

Рисунок 2.7 - Алгоритм декодування

Роботу даного алгоритму на основі стиснених даних, отриманих в результаті стиснення вище розглянутого рядка (/WED/WE/WEE/WEB/WET), демонструє рис. 2.8.

Вихідний потік точно відповідає вхідному потоку алгоритму стиснення. Перші 256 кодів визначені для переведення одиничних символів, так само як і в алгоритмі стиснення.

Вхідні коди: / W E D 256 E 260 261 257 B 260 T

Вхід Новий код	Старий код	Рядок Вихід	Символ	Новий вхід Таблиці
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

Рисунок 2.8 - Процес декодування

2.10 Арифметичне кодування

На відміну від префіксного кодування, арифметичне кодування дозволяє кодувати події з вірогідністю p кількістю біт, як завгодно близькою до величини $-\log_2 p$ [14].

Нехай необхідно закодувати послідовність подій, причому в кожен момент може наступити подія A, B або C. Відкладемо імовірність подій A, B і C на відрізку [0,1].

Робота алгоритму наочно показана на рис. 2.9, де на 4 діаграмах схематично відображені перші 3 кроки роботи алгоритму для вхідного алфавіту, що складається з подій “A”, “B” і “C” під час надходження на вхід алгоритму потоку подій “B”, “A”, “B”. Перша діаграма показує поділ інтервалу (0,1) на початкові підінтервали між символами вхідного алфавіту до початку роботи алгоритму. Після надходження на вхід алгоритму події “B” робочий інтервал (X,Y) аналогічним чином ділиться між символами вхідного алфавіту (діаграма 2) і т.д. Діаграма 4 на рис. 2.2 відповідає заштрихованій ділянці діаграми 1.

У міру кодування подій з вхідного потоку робочий інтервал (X,Y) (X’,Y’) (X’,Y’) ... скорочується. Наскільки він скоротиться при кодуванні однієї події, залежить від ширини початкового інтервалу, відповідного події (імовірність події).

Теоретично вихідними даними алгоритму може служити будь-яке число, що входить в робочий інтервал після обробки останньої події вхідного потоку. На практиці ж при кожному скороченні інтервалу у вихідний потік поміщаються всі збіжні старші біти верхньої і нижньої межі робочого інтервалу. Очевидно, що під час надходження події з більшою імовірністю і з ширшим початковим інтервалом робочий інтервал скоротиться менше і менше бітів потрапить у вихідний потік. Таким чином, події з більшою імовірністю появи кодуються меншим числом бітів.

Недоліком арифметичного кодера є його висока обчислювальна складність. Для кожного біта вихідного потоку необхідно виконувати ряд операцій порівняння і зсуву. Крім того, для кожної кодованої події необхідно виконати ряд операцій множення і ділення.

З метою підвищення швидкодії було запропоновано ряд швидших алгоритмів. Як найбільш швидка альтернатива арифметичному кодуванню

може бути використано квазіарифметичне [14]. У ньому операції множення і ділення замінені операціями вибору значення з таблиці. Алгоритм організований у вигляді кінцевого цифрового автомата. Під час надходження на вхід кодованої події алгоритм залежно від поточного стану при необхідності поміщає у вихідний потік деякі біти і переходить в деякий новий стан. Таблиці переходів і виходів складені так, щоб емулювати роботу арифметичного кодера. До недоліків квазіарифметичного кодера можна віднести низьку точність (або надмірно великий об'єм таблиць), а також те, що він реалізований тільки для випадку з двома подіями.

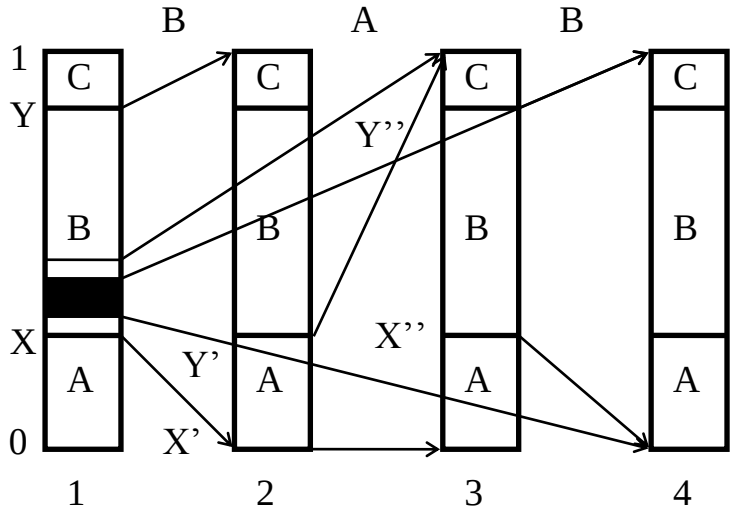


Рисунок 2.9 - Схема работи алгоритму арифметичного кодування

Вдалою альтернативою арифметичному кодеру, широко використовуваною в наш час, є Range-кодер [14]. Основною його відмінністю від арифметичного кодера є не побітне, а побайтне порівняння верхньої і нижньої межі і побайтне розміщення закодованого результату у вихідний потік. Такий підхід дозволяє помітно підвищити швидкодію кодера. При цьому точність (якість кодування) у окремих реалізацій Range-кодерів виявляється навіть вища, ніж у класичного арифметичного кодера (при використанні 32-бітової цілочислової арифметики).

Приклад роботи Range-кодера

Розглянемо процес арифметичного кодування слова "REDUNDANCE" (надмірність) [12]. Імовірність появи кожного символу в цьому слові дорівнює 0,1 за винятком букв E, D і N, що зустрічаються

двічі, і, відповідно, імовірність їхньої появи дорівнює 0,2. Далі кожній букві присвоюється інтервал імовірності (range), довжина якого розраховується, виходячи з імовірності їхньої появи в слові (табл. 2.11).

Перша буква слова – “R” - одержує інтервал з нижньою границею 0,8 і з верхньою – 0,9. Нижня границя інтервалу і стає першою значущою цифрою коду. Потім виконується розрахунок границь підінтервалів для кожної наступної букви за такими виразами:

$$\left.\begin{aligned} \beta_n^l &= \beta_{n-1}^l + (\beta_{n-1}^h - \beta_{n-1}^l)P_n^l \\ \beta_n^h &= \beta_{n-1}^l + (\beta_{n-1}^h - \beta_{n-1}^l)P_n^h \end{aligned}\right\}, \qquad (2.2)$$

де β^l, β^h – нижня і верхня границя кодового інтервалу;

P^l і P^h – нижня і верхня границі інтервалу імовірності для символу.

У результаті, послідовність символів 'REDUNDANCE' заміняється числом 0,8478570048, що показано в табл. 2.12. Таким чином, замість 10 байт, необхідних для збереження символьного рядка, буде потрібно всього 4 байти для запису числа.

Арифметичне кодування дозволяє забезпечити високий ступінь стиску даних, особливо у випадках, коли зустрічаються дані, де частота появи різних символів дуже відрізняється одна від одної. У той же час сама процедура арифметичного кодування потребує потужних обчислювальних ресурсів, і донедавна цей метод мало застосовувався при кодуванні зображень через повільну роботу алгоритму і, відповідно, істотного часу затримки при передачі даних. Хоча, варто очікувати високих коефіцієнтів стиску при його застосуванні для кодування зображень.

Таблиця 2.11 - Інтервали імовірності для символів в слові Redundance

Символ	Імовірність (p)	Інтервал
A	0,1	0,0-0,1
C	0,1	0,1-0,2
D	0,2	0,2-0,4
E	0.2	0,4-0,6
N	0,2	0,6-0,8
R	0,1	0,8-0,9
U	0,1	0,9-1,0

Таблиця 2.12 - Покрокове подання рядка REDUNDANCE методом арифметичного кодування

Символ	Нижня границя (β^l)	Верхня границя (β^h)
R	0,8	0,9
E	0,84	0,86
D	0,844	0,848
U	0,8476	0,848
N	0,84784	0,84792
D	0,847856	0,847872
A	0,8478560	0,8478576
N	0,84785696	0,84785728
C	0,847856992	0,847857024
E	0,8478570048	0,8478570432

2.11 Сучасні тенденції розвитку ущільнення даних. BWT-перетворення

З появою методу арифметичного кодування проблема генерації коду була фактично вирішена. З тих пір основна увага стала приділятися питанням, пов'язаним з моделюванням. Нові підходи опираються на парадигму ущільнення за допомогою універсального моделювання і кодування *(universal modelling and coding)*, запропоновану Ріссаненом і Ленгдоном (Langdon) в 1981 р. [10]. Відповідно до даної концепції процес ущільнення складається з двох самостійних частин:

- моделювання;
- кодування.

Під моделюванням розуміється побудова моделі інформаційного джерела, що породжує дані, які ущільнюються, а під кодуванням - відображення оброблюваних даних в стислу форму подання на підставі результатів моделювання (рис. 2.10).

"Кодувальник" створює вихідний потік, що є компактною формою подання оброблюваної послідовності, на підставі інформації, що поставляється йому "моделювальником".

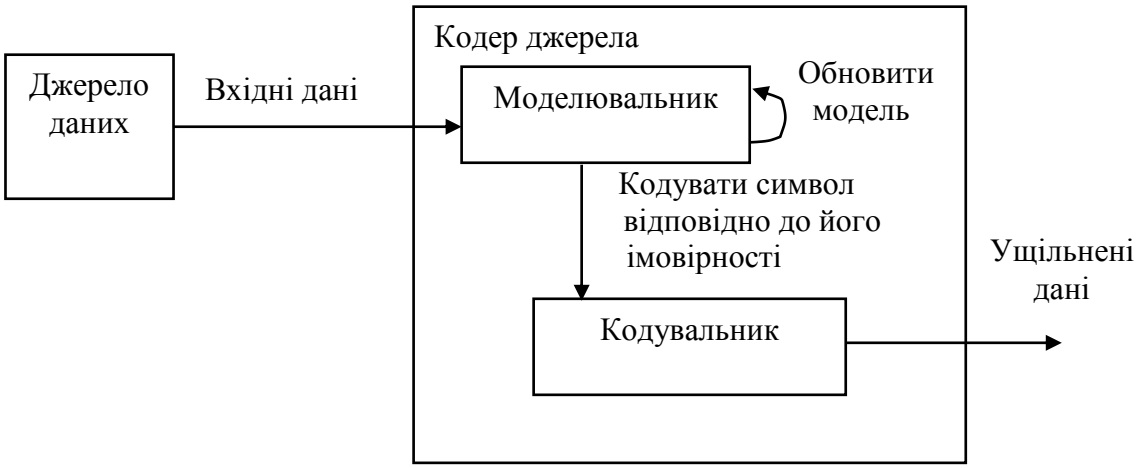


Рисунок 2.10 – Нова схема ущільнення даних

Слід відзначити, що поняття "кодування" часто використовують в широкому сенсі для позначення всього процесу ущільнення, тобто включаючи моделювання в даному нами означенні. Таким чином, необхідно розрізняти поняття кодування в широкому сенсі (весь процес) і у вузькому (генерація потоку кодів на підставі інформації моделі). Поняття "статистичне кодування" також використовується, часто з сумнівною коректністю, для позначення того або іншого рівня кодування. Щоб уникнути плутанини ряд авторів застосовує термін "ентропійне кодування" для кодування у вузькому сенсі. Це найменування далеке від досконалості і зустрічає цілком обґрунтовану критику. Далі в цьому розділі процес кодування в широкому сенсі іменуватимемо "кодуванням", а у вузькому сенсі - "статистичним кодуванням" або "власне кодуванням".

Оцінювання імовірності символів при моделюванні проводиться на підставі відомої статистики і можливо, апріорних припущень, тому часто говорять про завдання статистичного моделювання. Можна сказати, що моделювальник передбачає імовірність появи кожного символу в кожній позиції вхідного рядка, звідси ще одне найменування цього компонента - "передбачувач" або "предиктор" (від predictor). Чим точніша оцінка імовірності появи символів, тим більше коди відповідають оптимальним, тим краще ущільнення. На етапі статистичного кодування виконується заміщення символу s_i, з оцінкою вірогідності появи $p(s_i)$ кодом завдовжки $-\log_2 p(s_i)$ біт.

Виділяють 4 варіанти моделювання:

- статичне;
- напівадаптивне;
- адаптивне (динамічне);
- блоково-адаптивне.

При статичному моделюванні для будь-яких даних, що обробляються, використовується одна і та ж модель. Опис моделі зберігається в структурах кодера і декодера. Недоліком такого підходу є погане ущільнення або навіть збільшення даних, що не відповідають заданій моделі.

Найбільш простий спосіб оцінювання імовірностей реалізується за допомогою напівадаптивного моделювання і полягає в попередньому підрахунку частот появ символів в блоці даних, що ущільнюється.

При адаптивному моделюванні у міру кодування модель змінюється за заданим алгоритмом після ущільнення кожного символу. Однозначність декодування досягається тим, що спочатку кодер і декодер мають ідентичну і звичайно дуже просту модель, модифікація якої при кодуванні і декодуванні виконується однаковим чином. Коефіцієнти ущільнення при адаптивному моделюванні можуть навіть досягати значень близьких до напівадаптивного.

Блоково-адаптивне моделювання є окремим випадком адаптивної стратегії. В цьому випадку модель обновлюється не після обробки кожного символу, а після обробки блока символів. Довжина блока може змінюватись в процесі кодування.

Аналіз розповсюджених типів даних виявляє сильну залежність імовірності появи символів від попередніх символів у повідомленні, тобто більшість джерел повідомлень є джерелами з пам’яттю. Наприклад, для більшості європейських мов врахування безумовної імовірності символів дозволяє зменшити середню довжину коду до 4,5 біта на символ, а при використанні інформації про один попередній символ до 3,6 біта на символ. Врахування інформації про два попередні символи зменшує середню довжину коду до 3,2 біта на символ. Покращення ущільнення при врахуванні попередніх символів відзначається і для інших типів даних: об’єктних файлів, зображень, аудіоданих.

Моделювання, яке дає оцінку імовірності появи символу в залежності від попередніх, або контексту, називається контекстним

моделюванням. Контекстне моделювання практично завжди застосовується як адаптивне.

Контекст в широкому смислі - це сокупність символів, що оточують поточний символ. Розрізняють також лівосторонні і правосторонні контексти, тобто послідовності символів, що примикають до поточного символу зліва і справа, відповідно. При контекстному моделюванні під контекстом розуміють саме лівосторонній контекст, оскільки контекстне моделювання практично завжди застосовується як адаптивне.

Якщо довжина контексту обмежена, то такий підхід називається контекстним моделюванням обмеженого порядку (finite context modeling), при цьому під порядком розуміється максимальна довжина N контекстів, що використовуються.

Найбільш широке застосування отримала техніка контекстного моделювання PPM (Prediction by Partial Matching) – передбачення за частковим збігом та її модифікації. PPM забезпечує ущільнення в 3-4 рази для текстів і в 2-3 рази для об’єктних файлів при прийнятних обчислювальних затратах. Серед інших методів контекстного моделювання в широкому сенсі можна відзначити такі:

- моделі станів (використовується при динамічному марковському ущільненні – Dinamic Markov Compression або DMC);
- граматичні моделі (використовуються в алгоритмі SEQUITUR);
- моделі з використанням штучних нейронних мереж для побудови передбачувача [10].

Детальний розгляд методів контекстного моделювання виходить за межі даного навчального посібника.

В світлі концепції універсального моделювання і кодування заслуговують на увагу методи ущільнення без втрат на основі перетворень. Мета використання перетворень в ущільненні даних - перетворення потоку вхідних подій до вигляду, що дозволяє використовувати простіші і ефективніші моделі. Фактично вони перетворюють одні види надмірності в інші, простіше модельовані. Тобто, перетворення дозволяє подавати оброблювану інформацію в особливій формі, ідеально відповідній для подальшого ефективного кодування. Незвичність підходу полягає в наявності фактично двох етапів моделювання: перший етап - це робота перетворення, направлена на отримання «зручного» інформаційного

подання, а другий - побудова допоміжної моделі, на основі якої буде закодовано дане подання.

До таких перетворень відносять перетворення MTF (див. п. 2.8) та перетворення BWT (Burrows-Wheeler Transform) [10]. Однак якщо MTF давно використовується при ущільненні як перетворення, так і самостійний метод ущільнення, то по-перше перетворення BWT може використовуватись тільки як перетворення, а по-друге за рахунок використання перетворення BWT сумісно з MTF можна досягнути значних коефіцієнтів ущільнення. Перетворення BWT застосовується для перетворення ланцюжкової надмірності в надмірність повторення подій [10, 14]. Спочатку вхідний потік подій циклічно зсувається на одну позицію і записується під початковим вхідним потоком стільки раз, скільки подій у вхідному потоці. Отримана квадратна матриця сортується по рядках зліва направо. Доведено, що для відновлення початкового потоку подій достатньо останнього стовпця матриці (так званого префіксного стовпця) і номера рядка початкового потоку подій після сортування. Префіксний стовпець має велику надмірністю повторення подій і локальну надмірність розподілу імовірності.

Існують швидкі і ефективні реалізації даного перетворення, що не вимагають повної побудови квадратної матриці в пам'яті (що привело б до неприпустимо високих вимог до використовуваної пам'яті).

Вихідний потік перетворення BWT, як правило, або кодується за методом RLE, або додатково перетворюється за методом MTF і далі кодується або префіксним, або арифметичним кодером. Перетворення BWT використовується для стиснення текстової і графічної інформації. Розглянемо більш детально порядок виконання BWT-перетворення.

BWT-перетворення

Де-факто описувати BWT прийнято за допомогою прикладу перетворення рядка символів "абракадабра". Далі потрібно з рядка даних створити матрицю всіх можливих його циклічних перестановок. Першим рядком матриці буде початкова послідовність, другим рядком - вона ж, зсунута на один символ вліво, і т.д. Таким чином, отримаємо матрицю, зображену на рис. 2.11.

0 абракадабра
1 бракадабраа
2 ракадабрааб
3 акадабраабр
4 кадабраабра
5 адабраабрак
6 дабраабрака
7 абраабракад
8 браабракада
9 раабракадаб
10 аабракадабр

Рисунок 2.11- Матриця циклічних перестановок рядка "абракадабра”

Оскільки дані поступають з файла побайтно і якщо відомий розмір блока BWT-перетворення, то немає сенсу очікувати прийому всього блока, над яким виконується перетворення. Прийнятий байт спочатку записується в порядку прийому в “0” рядок матриці (рис. 2.11), а потім записується в інші рядки матриці в позиції, які визначаються за таким виразом:

$$Pos=(rbwt - ja + ia) \bmod rbwt,$$

де rbwt – розмір блока BWT-перетворення;

ia-номер поточного стовпця;

ja – номер рядка, в який записується черговий байт.

Наприклад, для рис. 2.8 - rbwt=11, нехай ja=0 і ia=6, і нехай ja=7 і ia=6, тоді:

0-рядок: Pos=(11 –0 +6) mod 11=6, тобто буква “д”;

7-рядок: Pos=(11 –7 +6) mod 11=10, тобто буква “д”.

Відсортуємо всі рядки даної матриці відповідно до лексикографічного порядку символів. Вважатимемо, що один рядок повинен знаходитися в матриці вище за інший в тому випадку, якщо в найлівішій з позицій, починаючи з якої рядки відрізняються, в цьому рядку знаходиться символ лексикографічно менший, ніж у іншого рядка. Іншими словами, слід відсортувати символи спочатку за першим символом, потім рядки, у яких перші символи однакові за другим символом і т.д. (рис. 2.12).

Тепер залишився останній крок - виписати символи останнього стовпця і запам’ятати номер початкового рядка серед відсортованих. Отже, "рдакрааааабб", 2 - це результат, отриманий в результаті перетворення Барроуза-Уілера.

0 аабракадабр
1 абраабракад
2 абракадабра - початковий рядок
3 адабраабрак
4 акадабраабр
5 браабракада
6 бракадабраа
7 дабраабрака
8 кадабраабра
9 раабракадаб
10 ракадабрааб

Рисунок 2.12 - Матриця циклічних перестановок рядка "абракадабра" відсортована зліва направо відповідно до лексикографічного порядку символів

Розглянемо процес відновлення початкової матриці. Нехай нам відомий тільки результат перетворення, тобто - "рдакрааааббб", 2. Відсортуємо всі символи останнього стовпця (рис. 2.13) відповідно до лексикографічного порядку. Очевидно, що в результаті такого сортування ми отримали перший стовпець початкової матриці. Оскільки останній стовпець відомий, додамо його в отриману матрицю (рис. 2.14).

0 а
1 а
2 а
3 а
4 а
5 б
6 б
7 д
8 к
9 р
10 р

Рисунок 2.13 - Відсортовані символи початкового рядка

0 а...р
1 а...д
2 а...а
3 а...к
4 а...р
5 б...а
6 б...а
7 д...а
8 к...а
9 р...б
10 р...б

Рисунок 2.14 - Перший і останній стовпці матриці циклічних перестановок

Тепер саме час пригадати, що рядки матриці були отримані в результаті циклічного зсуву початкового рядка. Тобто, символи останнього і першого стовпців утворюють один з одним пари. І нам ніщо не може перешкодити відсортувати ці пари, оскільки обов'язково існують такі рядки в матриці, які починаються з цих пар. І ще допишемо в матрицю і відомий нам останній стовпець (рис. 2.15).

0 аа...р
1 аб...д
2 аб...а
3 ад...к
4 ак...р
5 бр...а
6 бр...а
7 да...а
8 ка...а
9 ра...б
10 ра...б

Рисунок 2.15 - Перший, другий і останній стовпці матриці

Таким чином, два стовпці нам вже відомі. Легко помітити, що відсортовані пари разом із символами останнього стовпця складають трійки. Аналогічно відновлюється вся матриця. А на підставі записаного наперед номера початкового рядка в матриці - і сам початковий рядок (рис. 2.16).

0	**ааб… р**	**аабр...р**		**аабракада.р**	**аабракадабр**
1	**абр….л**	**абра...*д***		**абраабрак.д**	**абраабракад**
2	**абр… а**	**абра...а**		**абракадаб.а**	**абракадабра**
3	**ада…к**	**адаб...к**		**адабраабр.к**	**адабраабрак**
4	**ака… р**	**акад...р**		**акадабраа.р**	**акадабраабр**
5	**бра… а**	**браа...а**	**...**	**браабрака.а**	**браабракада**
6	**бра… а**	**брак...а**		**бракадабр.а**	**бракадабраа**
7	**даб… а**	**дабр...а**		**дабраабра.а**	**дабраабрака**
8	**кад… а**	**када...а**		**кадабрааб.а**	**кадабраабра**
9	**раа... б**	**рааб...б**		**раабракад.б**	**раабракадаб**
10	**рак... б**	**рака...б**		**ракадабра.б**	**ракадабрааб**

Рисунок 2.16 - Процес визначення всіх стовпців матриці

Однак такий підхід вимагає великих затрат пам’яті. Існує метод швидкого оберненого перетворення. Для швидкого оберненого перетворення додатково до власне даних потрібний вектор оберненого перетворення, що є масивом чисел, розмір якого рівний числу символів в блоці. Порядок отримання вектора оберненого перетворення може бути таким. Після отримання початкового рядка - "рдакрааааабб", 2, "рдакрааааабб" записується в три стовпці масиву, як показано на рис. 2.17.

0	**ррр**
1	**ддд**
2	**ааа**
3	**ккк**
4	**ррр**
5	**ааа**
6	**ааа**
7	**ааа**
8	**ааа**
9	**ббб**
10	**ббб**

Рисунок 2.17 - Три стовпці початкової матриці

0	**рра**	**0**
1	**дда**	**1**
2	**ааа**	**2**
3	**кка**	**3**
4	**рра**	**4**
5	**ааб**	**5**
6	**ааб**	**6**
7	**аад**	**7**
8	**аак**	**8**
9	**ббр**	**9**
10	**ббр**	**10**

Рисунок 2.18 – Чотири стовпці початкової матриці

Другий стовпець матриці відсортуємо в лексикографічному порядку і допишемо четвертий стовпець, який буде містити номер даного рядка (рис. 2.18). Нульовий і перший стовпець залишаються незмінними.

Відсортуємо рядки матриці за першим і другим стовпцями в лексикографічному порядку, нульовий стовпець матриці при цьому залишається незмінним. Результати наведено на рис. 2.19.

0	**раа**	**2**
1	**даб**	**5**
2	**ааб**	**6**
3	**кад**	**7**
4	**рак**	**8**
5	**абр**	**9**
6	**абр**	**10**
7	**ада**	**1**
8	**ака**	**3**
9	**бра**	**0**
10	**бра**	**4**

Рисунок 2.19 – Відсортовані рядки матриці

Останній стовпець чисел і є вектором оберненого перетворення. Тепер отримати початковий рядок зовсім просто. Насамперед візьмемо елемент вектора оберненого перетворення, відповідний номеру початкового рядка в матриці циклічних перестановок, T[2]=6. Інакше кажучи, як перший символ в початковому рядку слід узяти шостий символ з нульового стовпця "рдакрааааабб". Це символ "а". Далі T[6]=10. Це десятий символ з нульового стовпця "рдакрааааабб" - “б”. T[10]= 4 - “р”,

T[4]=8 - “а”, T[8]=3 - “к”, T[3]= 7 - “а”, T[7]= 1 - “д”, T[1]= 5 - “а”, T[5]= 9 - “б”, T[9]= 0 - “р”, T[0]= 2 - “а”. В результаті отримаємо слово “абракадабра”, що і потрібно.

2.12 Ущільнення зображень і звуку з втратами

Для ущільнення зображень і звуку (аудіо) можуть використовуватись методи ущільнення без втрат, розглянуті в даному розділі. Однак коефіцієнт ущільнення, який досягається при використанні цих методів, незначний – приблизно 1,5 – 2 рази.

Необхідно відзначити, що файли зображень і звуку характеризуються великими об’ємами, тому досягнення прийнятних коефіцієнтів ущільнення вкрай важливо. Крім того, зображення і звук в основному орієнтовані на сприйняття людиною. Це дає можливість створити спеціальні алгоритми ущільнення з втратами несуттєвої для людини інформації, які призначені тільки для зображень або звуку [12, 17-18].

Алгоритми ущільнення зображень, відео- та аудіоданих ґрунтується на наявності в цих даних надмірності з точки зору сприйняття людиною.

Так в основі більшості способів ущільнення звуку з втратами лежать психоакустичні особливості сприйняття звуку людиною "кодування для сприйняття" (perceptual coding), при якому із звукового сигналу видаляється інформація, малопомітна для слуху. В результаті, не дивлячись на зміну форми і спектра сигналу, його слухове сприйняття практично не змінюється, а ступінь стиснення виправдовує незначне зменшення якості. Таке кодування відноситься до методів стиснення з втратами (lossy compression), коли із ущільненого сигналу вже неможливо точно відновити початкову хвильову форму. Прийоми видалення частини інформації базуються на особливості людського слуху [18]:

- частотний спектр, що сприймається людиною (приблизно) від 20 Hz до 20 kHz, найбільша чутливість в діапазоні від 2 до 4 KHz;
- динамічний діапазон (від найтихіших звуків до найгучніших) близько 96 дб (більше ніж 1 до 30000 за лінійною шкалою);
- загальновідомо, що людина в змозі розрізнити зміну частоти на 0,3% на частоті порядку 1кГц;

- якщо два сигнали розрізняються менше ніж на 1дБ за амплітудою - їх важко розрізнити. Роздільна здатність за амплітудою залежить від частоти і найбільша чутливість спостерігається в діапазоні від 2 до 4 kHz;
- просторова роздільна здатність (здібність до локалізації джерела звуку) - до 1 градуса;
- звуки різної частоти розповсюджуються в повітрі з різною швидкістю. Високочастотна частина спектра від джерела, що віддалене від слухача, дещо запізнюється;
- людина не в змозі помітити раптове зникнення високих частот, якщо воно не перевищує порядку 2 мс;
- деякі дослідження показують, що людина в змозі відчувати частоти вищі 20 кГц;
- людська система сприйняття звуку має обмежену, залежну від частоти роздільну здатність. Рівномірна, з погляду сприйняття людиною, зміна частоти може бути виражене в одиницях ширини критичних смуг. Їх ширина менше 100 Гц для нижніх чутних частот, і більше 4 kHz для найбільш високих. Весь частотний діапазон може бути розділений на 25 критичних смуг.

Зображення також містить надлишкову інформацію, яку можна видалити при ущільненні. В зображенні розрізняють два основних види надмірності [12]:

- статистична надмірність;
- фізіологічна надмірність.

Перша пов'язана з тим, що будь-які величини, отримані із зображення не є випадковими. Сусідні відліки часто мають подібні значення яскравості, в чому проявляється важлива властивість їх просторової кореляції. Якщо відповідним чином використати цю властивість, то можна значно зменшити число біт для подання зображення у цифровій формі.

Фізіологічна надмірність пов'язана з тією частиною інформації, яка не сприймається оком людини. Скорочення фізіологічної надмірності значною мірою скорочує і статистичну надмірність і навпаки.

Детальний огляд особливостей зорового сприйняття наведений в [17], де виділені основні особливості зорового сприйняття зображень

корисні при розробці пристроїв кодування. Особливості зорового сприйняття нерухомих зображень наведені в табл. 2.13, де також вказаний спосіб застосування при стисненні.

Особливі можливості для ущільнення інформації є при компресії відео. У ряді випадків велика частина зображення передається з кадру в кадр без змін, що дозволяє будувати алгоритми ущільнення на основі вибіркового відстежування тільки частини «картинки» [12].

Таблиця 2.13 - Основні особливості зорового сприйняття

Вид зорового сприйняття	Як застосовувати	Спотворення
Сприйняття змін яскравості	Малі прирости, області чорного кодуються точніше; застосування логарифмічних шкал	Непомітні
Сприйняття просторових частот	Високочастотна складова кодується меншим числом розрядів	Непомітні
Сприйняття кольору	Перехід від моделі RGB до моделі YUV; сигнали кольоровості UV піддаються субдискретизації	Непомітні
Сприйняття підкреслення контурів	У місцях наявності контурів динамічний діапазон високочастотної компоненти збільшується	Непомітні, поліпшення візуальної якості зображення

Таким чином, загальна схема ущільнення зображення або звуку з втратами включає такі етапи.

1. На першому етапі знаходиться подання зображення або звуку у вигляді набору коефіцієнтів деякого математичного перетворення (наприклад, перетворення Фур’є, дискретне косинусне перетворення, Wavelet-перетворення або інше). Ця операція як правило обернена або умовно обернена.
2. На другому етапі зменшується точність подання компонент зображення або звуку (коефіцієнтів перетворення), але так, щоб виконувались задані вимоги до якості зображення або звуку. Така операція призводить до втрат інформації, тому не є оберненою.

3. На третьому етапі усувається статистична надмірність в зображенні, отриманому після виконання перших двох етапів. Для виконання цього етапу може застосовуватись кодування Хаффмана, арифметичне кодування та інші. Ця операція обернена [12].

Найбільш інтенсивні дослідження виконуються з пошуку нових методів для виконання першого і другого етапів, оскільки при цьому витрачається найбільше обчислювальних ресурсів і дослідження пов'язані не тільки з пошуком математичного перетворення, але і з дослідженням особливостей зорового сприйняття зображення і особливостей завадостійкої передачі цього зображення по каналах зв'язку.

Найвідомішими стандартами ущільнення зображень, відео і звуку є стандарти JPEG та MPEG [9, 12].

Кодування зображень і звуку вивчається в курсах, присвячених обробці сигналів і детально висвітлюється в багатьох публікаціях, наприклад в [17-18].

Контрольні питання і вправи

1. Які переваги застосування нерівномірних кодів?
2. Яка нерівність задає умову однозначного декодування нерівномірного коду?
3. Поясніть поняття миттєвого коду.
4. В яких випадках застосовують блокові укорочені коди?
5. Поясніть принцип побудови блокового укороченого коду.
6. Які ви знаєте нерівномірні коди?
7. Сформулюйте теорему Шеннона для кодування каналу без завад.
8. Наведіть класифікацію алгоритмів ущільнення без втрат.
9. Наведіть та охарактеризуйте метод ефективного кодування Шеннона-Фано.
10. Охарактеризуйте словникові методи ущільнення даних.
11. Охарактеризуйте статистичні методи ущільнення даних.
12. Дайте характеристику стиснення зображень методом RLE.
13. Які недоліки кодування Хаффмена?
14. Наведіть порядок побудови кодового дерева.
15. Що таке префіксні коди?

16. Наведіть алгоритм кодування методом LZW.
17. Наведіть алгоритм декодування методом LZW.
18. Поясніть основи арифметичного кодування.
19. Які основні недоліки арифметичного кодування?
20. Як працює Range – кодер?
21. Які недоліки алгоритму LZW?
22. Чи забезпечують алгоритми ущільнення без втрат високий коефіцієнт ущільнення зображень?
23. Який основний недолік арифметичних методів ущільнення даних?
24. На якому етапі кодування зображень застосовуються алгоритми ущільнення без втрат?

Вправи

1. Виконати ефективне кодування ансамблю із 4 знаків згідно з методом Шеннона-Фано, обчислити ентропію ансамблю та середню кількість бітів на знак повідомлення після кодування. Імовірності знаків такі: $p(z_1)=1/2$, $p(z_2)=1/3$, $p(z_3)=1/12$, $p(z_4)=1/12$. Відповідь: $H(Z)=1{,}63$ біт, $l_{ср}=1{,}67$ біт/знак.
2. Виконати ефективне кодування Шеннона-Фано блоками по 2 символи повідомлень, утворених за допомогою алфавіту, що складається з двох знаків z_1 і z_2 з імовірностями появи $p(z_1)=0{,}7$, $p(z_2)=0{,}3$. Обчислити ентропію повідомлення та середню кількість бітів на знак повідомлення після кодування. Відповідь: $H(Z)=0{,}88$ біт, $l_{ср}=0{,}9$ біт/знак.
3. Виконати ефективне кодування Шеннона-Фано блоками по 3 символи повідомлень, утворених за допомогою алфавіту, що складається з двох знаків z_1 і z_2 з імовірностями появи $p(z_1)=0{,}7$, $p(z_2)=0{,}3$. Обчислити ентропію повідомлення та середню кількість бітів на знак повідомлення після кодування. Відповідь: $H(Z)=0{,}88$ біт, $l_{ср}=0{,}9$ біт/знак.
4. Подати знаки повідомлення “абракадабра” префіксним кодом згідно з методом Хаффмана. Застосувати дерево коренем вверх, ліва гілка “0”, права “1”. Відповідь: а=1, б=01, р=001, к=0001, д=0000.
5. Виконати ефективне кодування повідомлення “ddaaaaaccc” методом кодування за ступенем новизни. Обчислити середню кількість бітів

на знак повідомлення після кодування. Відповідь: вихід - 11010100001100, $l_{ср}$=1,3 біт/знак.

6. Виконати ефективне кодування повідомлення “dbaaaaaccc” методом MTF. Обчислити ентропію повідомлення та середню кількість бітів на знак повідомлення після кодування. Відповідь: вихід - 111110110000011100, H(Z)=1,68 біт, $l_{ср}$=1,8 біт/знак.
7. Ущільнити методом LZW таке повідомлення: “dedecdedec”. Відповідь: d e 256 c 256 258.
8. Ущільнити методом LZW таке повідомлення: “/we/we/wet”. Відповідь: / w e 256 258 257 t.
9. Виконати арифметичне кодування такого повідомлення “аббат”. Відповідь: 0.24448.
10. Виконати арифметичне кодування такого повідомлення “bedkmdomce”. Відповідь: 0.0478570048.
11. Виконати BWT-перетворення такої послідовності: “dedecdedec”. Відповідь: eeeeccdddd, 4.
12. Виконати BWT- перетворення такої послідовності: “abaka”. Відповідь: kabaa, 1.
13. Виконати обернене BWT-перетворення такої послідовності: тббаа, 0. Відповідь: аббат.
14. Розробити програму ущільнення файлів імовірнісним методом. Імена файлів вводяться з командного рядка. Мова програмування C++.
15. Розробити програму ущільнення файлів методом Шеннона-Фано. Мова програмування C++ .
16. Розробити програму ущільнення файлів напівадаптивним методом Хаффмана. Мова програмування C++ .
17. Розробити програму ущільнення файлів методом MTF. Мова програмування C++ .
18. Розробити програму ущільнення файлів напівадаптивним методом LZW. Мова програмування C++ .
19. Розробити програму ущільнення файлів з використанням арифметичного range-кодера. Мова програмування C++ .
20. Розробити програму виконання BWT-перетворення над файлами. Розмір блоків повинен задаватися користувачем в межах 8-256. Мова програмування C++.

3 ЗАВАДОСТІЙКЕ КОДУВАННЯ ІНФОРМАЦІЇ

3.1 Класифікація завад

Реальні системи передачі даних не ідеальні. Застосовуючи інформаційну техніку, ми повинні враховувати можливість виникнення помилок (імовірність помилок) при передачі і зберіганні інформації. Це в першу чергу відноситься до [19]:

- зберігання інформації на носіях з високою щільністю запису (магнітні носії, *CD -ROM, DVD);*
- передачі даних при обмеженій потужності сигналу (супутниковий і мобільний зв'язок);
- передачі інформації по дуже “зашумлених” каналах (мобільний зв'язок, високошвидкісні проводові лінії зв'язку);
- каналів зв'язку з підвищеними вимогами до надійності інформації (обчислювальні мережі, лінії передачі із ущільненням даних).

Причиною помилок є завади *(noise),* що діють в інформаційній системі [5].

Завада - сторонній вплив, що діє в системі передачі і заважає правильному прийому сигналів. Джерела завад можуть знаходитись як зовні, так і всередині самої системи передачі.

Якщо завада регулярна і відома, то боротьба з нею не викликає ускладнень. Однак, ситуація ускладнюється, якщо завади носять випадковий характер.

В загальному випадку дія завади ξ на сигнал, що передається, може бути виражена таким оператором [5]:

$$x = V(s, \xi), \tag{3.1}$$

де s – повідомлення, яке передається;

ξ – завада;

x – прийняте повідомлення.

В окремому випадку, якщо оператор V вироджується в суму

$$x = s + \xi, \tag{3.2}$$

то завада, яка діє в каналі, називається адитивною. Адитивну заваду часто називають шумом.

А якщо ж оператор V може бути поданий у вигляді

$$x = \nu \cdot s \ , \qquad (3.3)$$

де випадковий процес ν(t) невід'ємний, то заваду ν називають мультиплікативною.

Якщо ν змінюється повільно у порівнянні з s(t), то явище, викликане мультиплікативною завадою, називають *завмиранням каналу (федінг)*.

В більш загальному випадку оператор V не може бути приведений до основних форм (3.2), (3.3). При одночасній наявності шуму і мультипликативної завади зручно ввести два випадкових процеси, тобто оператор V подається так:

$$x = \nu \cdot s + \xi. \qquad (3.4)$$

З фізичної точки зору випадкові завади породжуються різного роду флуктуаціями. Флуктуаціями у фізиці називають випадкові відхилення тих або інших випадкових величин від їх середніх значень. Так, джерелом шуму в електричних ланцюгах постійного струму можуть бути флуктуації струму навколо середнього значення, які викликані дискретною природою носіїв зарядів (іонів і електронів). Це явище носить назву дробового ефекту.

Найбільш універсальною причиною шуму є флуктуації, обумовлені тепловими рухами. Випадковий тепловий рух носіїв заряду в будь-якому провіднику викликає випадкову різницю потенціалів на його кінцях. Ця різниця потенціалів флуктує навколо середнього значення рівного нулю, її середній квадрат пропорційний абсолютній температурі. Завада, що виникає, називається тепловим шумом.

Є ще одне джерело принципово неусовного шуму, яке викликано дискретною природою електромагнітного випромінювання. Як відомо, випромінювання виконується дискретними порціями – квантами, які називають фотонами. При високих частотах шуми, викликані дискретною фотонною структурою випромінювання, можуть перевищити всі інші завади.

Природа мультпликативної завади полягає у випадкових змінах параметрів каналу передачі. Причому ці зміни можуть носити як лінійний, так нелінійний характер.

Таким чином, флуктуації і обумовлені ними завади, є природним результатом дискретної природи ряду фізичних явищ і статистичної природи ряду фізичних величин.

Одним із способів боротьби з випадковими завадами в системах передачі та обробки інформації є застосування кодів, що контролюють помилки. Теорія завадостійкого кодування для кожного конкретного каналу дозволяє вибрати найбільш ефективний метод виявлення і виправлення помилок. Існують два взаємодоповнювальних методи боротьби з завадами [19]:

- кодування з виправленням помилок (коректуючі коди) - приймач виявляє і виправляє помилки;
- кодування з виявленням помилок - приймач розпізнає помилки і, у разі потреби, проводить запит на повторну передачу помилкового блока.

Останній метод припускає наявність каналу зворотного зв'язку і знаходить своє застосування в каналах з достатньо малою імовірністю помилки у випадку, якщо цю імовірність помилки необхідно ще знизити. Така ситуація часто виникає в обчислювальних мережах і в інтернеті. Типове значення імовірності помилки на біт без кодування в обчислювальних мережах складає 10^{-6}. Використання простих кодів з невеликою надмірністю дозволяє досягти вірогідності 10^{-9} і нижче. Вимога до імовірності помилки 10^{-9} не є надмірно завищеною. У обчислювальних мережах, наприклад, може виникнути обрив зв'язку в результаті пошкодження оптоволокна при проведенні земляних робіт, недбалого підключення кабелю до модему і т.д. Такий обрив повинен бути швидко виявлений декодером, який у разі різкого зростання частоти запитів на повторну передачу видає сигнал обриву зв'язку.

Теорія завадостійкого кодування базується на результатах досліджень, проведених американським вченим К. Шенноном [1].

3.2 Теорема Шеннона про кодування для каналу із завадами

Теорема Шеннона задає умови, при яких можлива передача інформації по каналу зв’язку із завадами з як завгодно малою імовірністю помилки. На цій теоремі ґрунтується теорія завадостійкого кодування.

Формулювання теореми таке [1].

1. При будь-якій продуктивності джерела інформації, меншій ніж пропускна здатність каналу, існує такий спосіб кодування, який дозволяє забезпечити передачу інформації, що створюється джерелом повідомлень з як завгодно малою ймовірністю помилки.
2. Не існує способу кодування, який дозволив би вести передачу інформації з як завгодно малою ймовірністю помилки, якщо продуктивність джерела повідомлення більша пропускної здатності каналу.

Теорема встановлює теоретичну межу можливої ефективності системи при достовірній передачі інформації. Теорема неконструктивна, в тому смислі, що в ній не торкаються питань про шляхи побудови кодів, які забезпечують вказану ідеальну передачу. Однак, обґрунтувавши принципову можливість такого кодування, вона мобілізувала зусилля вчених на розробку конкретних кодів.

3.3 Основні принципи завадостійкого кодування

Кодування з виявленням і виправленням помилок є метод обробки повідомлень, призначений для підвищення надійності передачі по цифрових каналах. Ідея виявлення помилок дуже проста. Для передачі використовуються не всі $N_0=m^n$ комбінацій, де m - основа коду, а n – значність (довжина) коду, а лише N комбінацій, причому $N<N_0$ [5].

Комбінації, які використовуються в даному коді, називаються *дозволеними*, а комбінації, що не використовуються, називаються *забороненими*. Якщо сукупність помилок в дозволеній кодовій комбінації перетворює її в іншу дозволену кодову комбінацію, то такі помилки не можуть бути виявлені.

Інтуїтивно зрозуміло, що чим більша відстань між дозволеними кодовими комбінаціями, тим більше помилок виправляє код. Для визначення відстані між двома кодовими комбінаціями прийнято використовувати таку метрику [4-5]:

$$d_{ij} = \sum_{k=1}^{n} \left| x_{ik} - x_{jk} \right|. \qquad (3.5)$$

Визначену таким чином відстань часто називають відстанню Хеммінга. Для двійкового коду відстань просто дорівнює кількості знаків в яких одна кодова комбінація відрізняється від іншої. Зазвичай її визначають шляхом додавання за модулем 2 двох кодових комбінацій і підрахунку кількості одиниць в цій сумі.

Найменшу відстань для даного коду називають *кодовою відстанню* і позначають символом "d_{min}".

Приклад 3.1. Визначити кодову відстань d_{min} для наступного коду: A_1=0000, A_2=0011, A_3=1100, A_4=1010.

Розв'зування. Знайдемо суми за модулем 2 всіх можливих кодових комбінацій і значення відстані між ними як кількість одиниць в отриманій сумі:

0000	0000	0000	0011	0011	1100
+	+	+	+	+	+
0011	1100	1010	1100	1010	1010
———	———	———	———	———	———
0011	1100	1010	1111	1001	0110

$d_{12} = d_{21} = 2$ $d_{13} = d_{31} = 2$ $d_{14} = d_{41} = 2$ $d_{23} = d_{32} = 4$ $d_{24} = d_{42} = 2$ $d_{34} = d_{43} = 2$

Таким чином, найменшим значенням d_{ij} є 2, тобто кодова відстань для даного коду така: d_{min}=2.

Для виявлення всіх помилок кратності q_d в кодовій комбінації необхідно і достатньо, щоб кодова відстань задовольняла таку нерівність [5]:

$$d_{min} \geq q_d + 1. \qquad (3.6)$$

Під кратністю помилок розуміють кількість неправильно переданих символів в межах однієї кодової комбінації. Тобто, мінімальне значення d_{min}=2.

Для виправлення помилок кратності q_c необхідно і достатньо щоб кодова відстань задовольняла таку нерівність:

$$d_{min} \geq 2q_c + 1. \qquad (3.7)$$

Тобто, мінімальне значення d_{min}=3.

Для виправлення помилок кратності q_c і виявлення помилок кратності q_d необхідно і достатньо щоб [5]:

$$d_{min} \geq q_c + q_d + 1, \quad (3.8)$$

причому $q_d \geq q_c$.

Наведені вирази дозволяють будувати прості завадостійкі коди.

Хоча різні схеми кодування дуже несхожі одна на одну і основані на різних математичних теоріях, всім їм притаманні дві загальні властивості.

Перша - використання надмірності. Закодовані послідовності завжди містять додаткові, або надмірні, символи. Кількість символів в кодовій послідовності завжди більша, ніж необхідно для однозначного подання будь-якого повідомлення.

Друга - властивість усереднювання, це означає, що надмірні символи залежать від декількох інформаційних символів, тобто інформація, що міститься в кодовій послідовності, перерозподіляється також і на надмірні символи [19].

3.4 Класифікація завадостійких кодів та їх основні параметри

Всі коди, які виявляють і виправляють помилки, можна розділити на *імовірнісні* та *алгебраїчні* [2]. В імовірнісних кодах здатність виявляти і виправляти помилки ґрунтується на визначені імовірності появи символів (наприклад, в кодах Вагнера). Однак вони не знайшли широкого застосування, тому в подальшому не розглядаються.

Розглянемо основні підходи до класифікації алгебраїчних кодів, яку наведено на рис. 3.1 [20].

Алгебраїчні коди характеризуються тим, що здатність виявляти і виправляти помилки є властивістю їх алгебраїчної структури. Алгебраїчні завадостійкі коди можна розділити на *блокові* та *деревовидні*.

В блокових кодах всі операції кодування і декодування виконуються над блоками фіксованої довжини. Кодер для блокового коду є пристроєм без пам’яті, який відображає послідовність з k символів в послідовність із n символів. Термін “без пам’яті” указує, що кожний вихідний блок з n символів залежить тільки від відповідного блока з k символів і не залежить від інших блоків. Основними параметрами блокового коду є довжина коду n, довжина інформаційної послідовності k, швидкість коду r=k/n і мінімальна кодова відстань d_{min}.

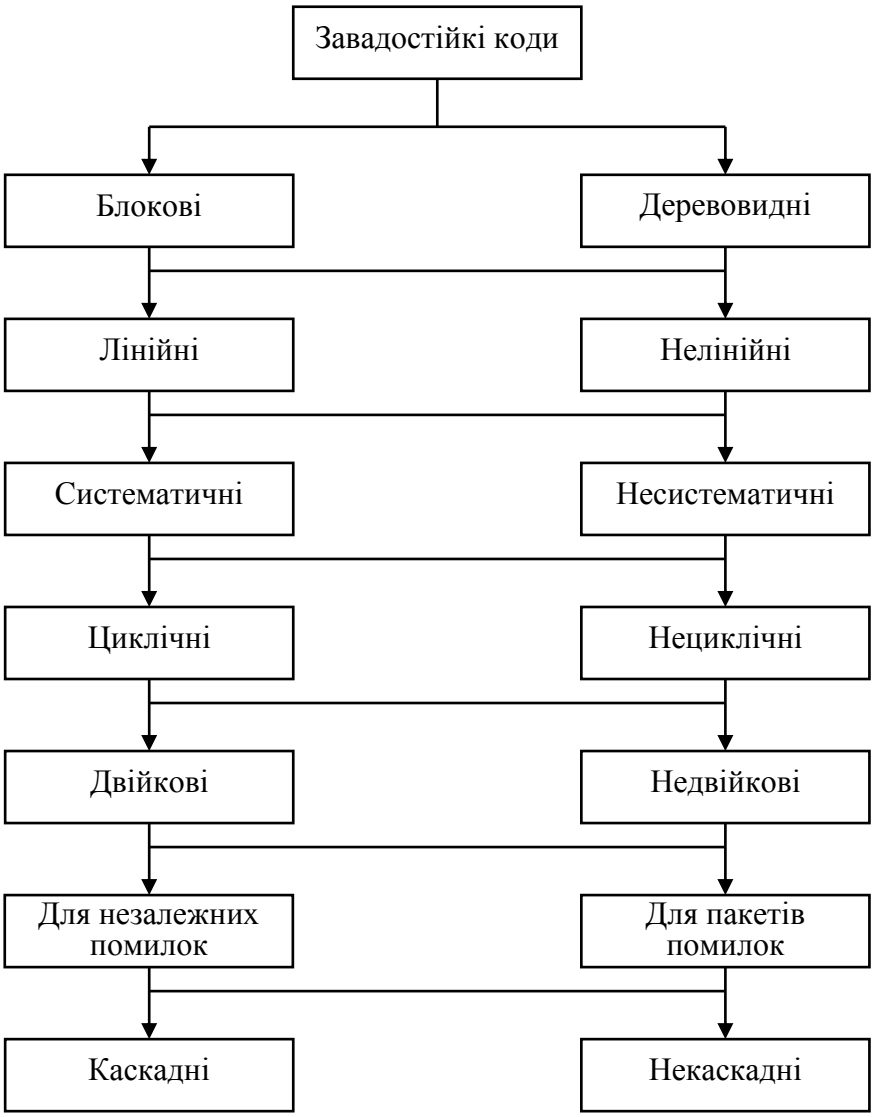


Рисунок 3.1 – Класифікація завадостійких кодів

В деревовидних кодах обробляється неперервна послідовність символів без розподілу її на блоки. Лінійні деревовидні коди називають згортними кодами. Вони є найпростішими з точки зору реалізації і тому знайшли найбільш широке застосування. Кодер для згортного коду є пристроєм з пам'яттю, в який поступають набори з k_0 вхідних символів, а на виході з'являються набори з n_0 вихідних символів. Кожен набір n_0 вихідних символів залежить від поточного вхідного набору і від K-1 попередніх вхідних наборів. Параметр K називається *конструктивною довжиною* згортного коду, а величину $n_A=Kn_0$ - *довжиною кодового обмеження.* Деревовидні коди характеризуються також швидкістю $r=k/n_0$ і

вільною відстанню d_{free}, сенс якої буде пояснений далі при описі згортних кодів.

Залежно від кількості можливих значень m *(основи коду)* кожного з символів всі коди можна розділити на *двійкові* (при m=2) і *недвійкові* (при m>2).

Можна також виділити *роздільні* коди і *нероздільні*. В *роздільних* кодах можна чітко визначити інформаційну частину і перевірні символи. В *нероздільних* кодах це зробити неможливо.

Особливістю лінійних кодів є те, що перевірні символи отримують шляхом лінійної комбінації інших символів коду. Кожний лінійний код має еквівалентний систематичний код. В систематичному коді перших k символів – інформаційні символи, а наступні – перевірні.

Важливий підклас лінійних кодів складають *циклічні* коди, які характеризуються циклічними властивостями. Якщо $\vec{C} = (c_0, c_1, ..., c_{n-1})$ - кодове слово циклічного коду, то і кодове слово $\vec{C}' = (c_1, ..., c_{n-1}, c_0)$, яке отримали шляхом циклічного зсуву елементів $\vec{C}$, також є кодовим словом даного коду. Ці коди мають ряд структурних особливостей, які спрощують реалізацію операцій кодування та декодування.

Відомі коди, що виправляють *випадкові* або *незалежні* помилки і коди, що виправляють *пакети* помилок. На практиці в основному використовуються коди, що виправляють *випадкові* помилки, оскільки для виправлення пакетів легше використати коди для виправлення випадкових помилок разом з пристроями перемішування і відновлення. Перший з них змінює порядок символів в закодованій послідовності, а другий відновлює початковий порядок після прийому. Тоді пакети помилок перед декодуванням будуть розбиті на випадкові помилки.

Важливим етапом в розвитку теорії кодування є поява каскадних кодів, в основі побудови яких лежить ідея сумісного використання декількох кодів. Дані джерела спочатку кодуються зовнішнім кодом, а потім закодовані символи зовнішнього коду кодуються кодером внутрішнього коду. Цей підхід дозволив істотно підвищити ефективність застосування кодування в порівнянні з базовими некаскадними методами. Мінімальна відстань сформованого каскадного коду буде рівна $D=d_1 \cdot d_2$, де d_1 і d_2 - мінімальні відстані складових кодів.

Майже всі схеми кодування, які застосовуються на практиці, ґрунтуються на лінійних кодах, тому розглянемо їх більш детально.

Таким чином, оскільки здатність виявити і виправити помилки досягається шляхом введення надмірності, то можна характеризувати завадостійкі коди за такими параметрами.

1. Довжина коду – n;
2. Довжина інформаційної послідовності - k;
3. Довжина перевірної послідовності - m=n-k;
4. Кодова відстань коду - d_{min};
5. Швидкість коду - r=k/n;
6. Надмірність коду –1/r;
7. Імовірність виявлення помилки - $p_{вп}$;
8. Імовірність не виявлення помилки (спотворення) - $p_{нв}$.

3.5 Граничні співвідношення між параметрами завадостійких кодів

Одним з найважливіших завдань побудови завадостійких кодів із заданими характеристиками є встановлення співвідношення між його здатністю виявляти або виправляти помилки і надмірністю. Існують граничні оцінки, які пов’язують d_{min}, n і k. Деякі з них такі [4, 8].

1. Межа Хеммінга, яка близька до оптимальної для високошвидкісних кодів, визначається співвідношеннями:

$$n - k \geq \log_q \sum_{i=0}^{t_u} C_n^i \cdot (q-1)^i, \tag{3.9}$$

де q- основа коду;

C_n^i - кількість сполучень з n елементів по i:

$$C_n^i = \frac{n!}{i! \cdot (n-i)!}. \tag{3.10}$$

Для двійкових кодів (q=2):

$$n - k \geq \log_2 \sum_{i=0}^{t_u} C_n^i. \tag{3.11}$$

2. Межа Плоткіна, яку доцільно використовувати для низькошвидкісних кодів визначається співвідношеннями: для q-ного коду:

$$d_0 \le n \cdot (q-1) \cdot q^{k-1} / (q^k - 1). \qquad (3.12)$$

Для двійкового коду:

$$d_0 \le n \cdot 2^{k-1} / (2^k - 1). \qquad (3.13)$$

Границі Хеммінга і Плоткіна є верхніми межами для кодової відстані при заданих n і k, задаючи мінімальну надмірність, при якій існує завадостійкий код, що має мінімальну кодову відстань і гарантовано виправляє t_u-кратні помилки.

3. Межа Варшамова-Гільберта (нижня межа), визначається такими співвідношеннями:

$$q^{n-k} > \sum_{i=0}^{d_{min}-2} C_{n-1}^{i} (q-1)^i; \qquad (3.14)$$

для двійкових кодів:

$$2^{n-k} > \sum_{i=0}^{d_{min}-2} C_{n-1}^{i}. \qquad (3.15)$$

Показує, при якому значенні n-k визначено існує код, що гарантовано виправляє помилки кратності t_u.

Для прикладу доведемо співвідношення (3.11). Нехай об’єм алфавіту джерела повідомлень N. Тоді кількість інформаційних біт, необхідна для подання символу алфавіту, така:

$$k = \log_2 N. \qquad (3.16)$$

Нехай відома кількість помилок *E*, які необхідно виправити. Загальна кількість помилкових комбінацій складе:

$$E \cdot 2^k = E \cdot N. \qquad (3.17)$$

Оскільки кількість заборонених комбінацій коду дорівнює $N_0 - N$, де $N_0 = 2^n$, n – значність коду, що виявляє і виправляє помилки, то необхідною умовою можливості виправлення помилок буде:

$$NE \le N_0 - N. \qquad (3.18)$$

Звідки:

$$\frac{N_0}{N} \ge 1 + E. \qquad (3.19)$$

А з урахуванням того, що $N_0=2^n$ і $N=2^k$, отримаємо:

$$2^{n-k} \geq 1+E. \tag{3.20}$$

Очевидно, що якщо t_u максимальна кратність помилок, які повинен виправляти шуканий код, то загальна кількість помилок для кратності від 1 до t_u така:

$$E=\sum_{i=1}^{t_u} C_n^i. \tag{3.21}$$

Оскільки $C_n^0=1$, то

$$1+E=\sum_{i=0}^{t_u} C_n^i. \tag{3.22}$$

Тоді з (3.19) одержимо:

$$2^{n-k} \geq \sum_{i=0}^{t_u} C_n^i. \tag{3.23}$$

Візьмемо логарифм за основою "2" від останнього виразу:

$$n-k \geq \log_2 \sum_{i=0}^{t_u} C_n^i. \tag{3.24}$$

Що і потрібно було довести.

3.6 Способи подання лінійних блокових кодів

Майже всі схеми кодування, які застосовуються на практиці, основані на лінійних кодах [2, 4, 8, 19-26], тому детальніше розглянемо спосіб їх опису. Блоковий код довжиною n з 2^k словами називається лінійним (n, k) кодом, якщо його кодові слова утворюють k-вимірний підпростір V_k векторного n-вимірного простору V_n. Підпростір V_k породжується базисом з k лінійно незалежних векторів, які утворюють рядки породжувальної матриці (n, k) коду:

$$G=\begin{bmatrix} g_0 \\ g_1 \\ \cdot \\ \cdot \\ \cdot \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots g_{1,n-1} \\ \cdots & \cdots & \cdots \\ g_{k,0} & g_{k,1} & \cdots g_{k-1,n-1} \end{bmatrix}. \tag{3.25}$$

Процес кодування блокового коду полягає в розбитті інформаційної послідовності на повідомлення $\vec{U} = (u_0, u_1, ..., u_{k-1})$ довжини k і відображенні цих повідомлень в кодові слова $\vec{C} = (c_0, c_1, ..., c_{n-1})$ довжини n таким чином:

$$\vec{C} = \vec{U}G. \tag{3.26}$$

Часто кодові слова краще подавати в систематичній формі $\vec{C} = (\vec{U}, \vec{V})$, утворюючи окремо інформаційну частину $\vec{U}$ із k символів і перевірну частину $\vec{V}$ із m=n-к символів. Породжувальна матриця такого систематичного коду матиме вигляд:

$$G = [I_k P] = \begin{bmatrix} 1 & 0 & ...0 & g_{0,0} & g_{0,1} & ...g_{0,n-1} \\ 0 & 1 & ...0 & g_{1,0} & g_{1,1} & ...g_{1,n-1} \\ ... & ... & ... & ... & ... & \\ 0 & 0 & ...1 & g_{k,0} & g_{k,1} & ...g_{k-1,n-1} \end{bmatrix}. \tag{3.27}$$

В даному випадку матриця *G* містить одиничну матрицю I_k розміром k x k, яка формує інформаційну частину слова, і матрицю P розміром k x (n-k), що визначає перевірні символи.

З породжувальною матрицею тісно пов'язано поняття перевірної матриці H, простір рядків якої ортогональний простору рядків породжувальної матриці, тобто

$$GH^T = 0. \tag{3.28}$$

Відзначимо, що кожне кодове слово лінійного коду $\vec{C}$ також задовольнятиме умову ортогональності, оскільки

$$\vec{C}H^T = \vec{U}GH^T = 0. \tag{3.29}$$

Якщо породжувальна матриця, задана у формі (3.27), то для виконання умови ортогональності перевірна матриця H повинна мати вигляд

$$H = [P^T I_{n-k}]. \tag{3.30}$$

Перевірна матриця використовується в процесі декодування коду таким чином. Нехай при передачі по каналу через вплив шумів $\vec{E}$ кодові символи $\vec{C}$ спотворюються. В результаті прийняті слова мають вигляд

$\vec{Y} = \vec{C} + \vec{E}$, де $\vec{Y} = (y_0, y_1, ..., y_{n-1})$ - прийнятий вектор, а $\vec{E} = (e_0, e_1, ..., e_{n-1})$ - вектор помилок. Для виявлення або виправлення помилок на приймальній стороні спочатку обчислюють синдром:

$$\vec{S} = \vec{Y}H^T = (s_0, s_1, ...s_{n-k-1}). \quad (3.31)$$

Синдром залежить тільки від вектора помилок, оскільки:

$$\vec{S} = \vec{Y}H^T = (\vec{C} + \vec{E})H^T = \vec{C}H^T + \vec{E}H^T. \quad (3.32)$$

Якщо $\vec{S} = 0$, то прийняте слово $\vec{Y}$ належить до множини кодових слів, інакше слово $\vec{Y}$ містить помилки, і за значенням символів синдрому $\vec{S}$ визначають їх місцеположення. Даний принцип лежить в основі методів синдромного декодування.

Важливий підклас лінійних кодів складають циклічні коди, які мають циклічні властивості: якщо $\vec{C} = (c_0, c_1, ..., c_{n-1})$ - кодове слово циклічного коду, тоді і $\vec{C}' = (c_1, c_2, ..., c_{n-1}, c_0)$, отримане циклічним зсувом елементів $\vec{C}$, також є кодовим словом. Ці коди мають ряд структурних особливостей, які значно спрощують реалізацію операцій кодування і декодування. Породжувальну матрицю циклічного коду можна записати у вигляді:

$$G = \begin{bmatrix} g_0 & g_1 & ...g_{n-k} & 0 & ... & 0 \\ 0 & g_0 & g_1 & ...g_{n-k} & ...0 & \\ ... & ... & ... & ... & ... & \\ 0 & ... & 0 & g_0 & g_1 & ... & g_{n-k} \end{bmatrix}. \quad (3.33)$$

Для компактного запису циклічного (n,k) коду часто використовується породжувальний або твірний поліном g(x) степеня n-k, з коефіцієнтами з поля GF(q) (для двійкових кодів q={0,l}):

$$g(x) = g_0 + g_1x + g_2x^2 + ... + g_{n-k}x^{n-k}. \quad (3.34)$$

Якщо у вигляді полінома записати інформаційне повідомлення U(x) і кодове слово C(x), то кодові слова коду утворюються шляхом множення повідомлення на породжувальний поліном:

$$C(x) = U(x)g(x). \quad (3.35)$$

Для зручності запису коефіцієнти твірних поліномів об'єднують в двійкове слово і подають у восьмеричній системі числення. Також твірні поліноми можна записувати шляхом перерахування степенів з ненульовими коефіцієнтами. Наприклад, запис $g(x) = 1 + x + x^4 + x^6$ рівносильний такому: $g = 1010011_2 = 123_8$; g = (0,1,4,6).

Для прикладу, розглянемо систематичний блоковий код, що відповідає такій породжувальній матриці:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \qquad (3.36)$$

Перевірна матриця даного коду відповідно до (3.30) записується таким чином:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.37)$$

Структурна схема даного кодера наведена на рис. 3.2. Отриманий код характеризується такими параметрами: кількість кодових символів n=6, кількість інформаційних символів k=3, кодова швидкість r=1/2, мінімальна кодова відстань $d_{min}=3$.

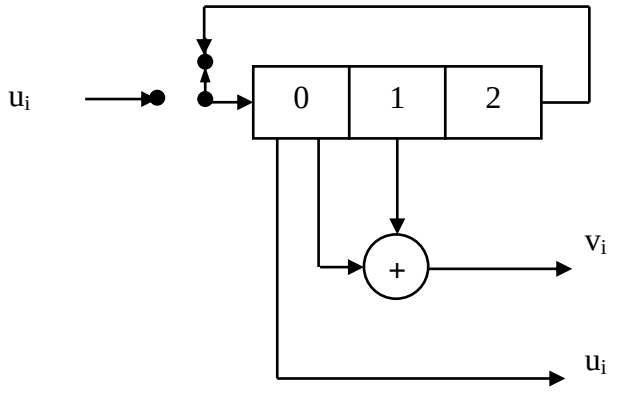


Рисунок 3.2 – Кодер блокового систематичного коду

3.7 Лінійні блокові коди

До лінійних кодів відносяться як двійкові *(binary codes)*, так і недвійкові коди. Розглянемо лише двійкові коди.

Блоковий код довжиною n символів, що складається з 2^k кодових слів, називається лінійним (n, k) –кодом, якщо всі його 2^k кодових слів утворюють k-вимірний підпростір векторного простору n - послідовностей двійкового поля GF(2).

Полем називається множина математичних об'єктів, які можна дадавати, віднімати, множити і ділити. Поле, що містять кінцеве число елементів називається полем Галуа. Візьмемо просте поле, що складається з двох елементів - нуля, - 0 і одиниці - 1. Визначимо для нього операції додавання і множення:

0 + 0=0, 0 · 0=0;
0 + 1=1, 0 · 1=0;
1 + 0=1, 1 · 0=0;
1 + 1=0, 1 · 1=1.

Визначені таким чином операції додавання і множення називаються додаванням за модулем 2 (mod 2) і множенням за модулем 2. Відзначимо, що з рівності 1+1 = 0 витікає, що –1=1 і відповідно 1+1=1-1**,** а з рівності 1· 1=1 - що 1:1=1.

Алфавіт з двох символів 0 і 1 разом з операціями додаванням і множенням по mod 2 називається полем з двох елементів, яке є найпростішим полем Галуа і позначається як GF(2).

До поля GF(2) можна застосувати всі методи лінійної алгебри, зокрема матричні операції. Всі дії над символами в двійкових кодах виконуються за модулем 2.

Таким чином, двійковий код є лінійним, якщо сума за модулем 2 двох кодових слів також є кодовим словом цього коду. Лінійний двійковий код є також груповим, оскільки сукупність кодових комбінацій, що входять в нього, утворюють групу. Група - множина математичних об'єктів, які можна додавати, віднімати [25].

Бажаною якістю лінійних блокових кодів є систематичність. Систематичний код має формат, зображений на рис. 3.3, тобто містить

незмінну інформаційну частину завдовжки k символів і надмірну (перевірну), довжиною n-k символів. Блоковий код, що має властивості лінійності і систематичності називається лінійним блоковим систематичним (n, k) –кодом [25].

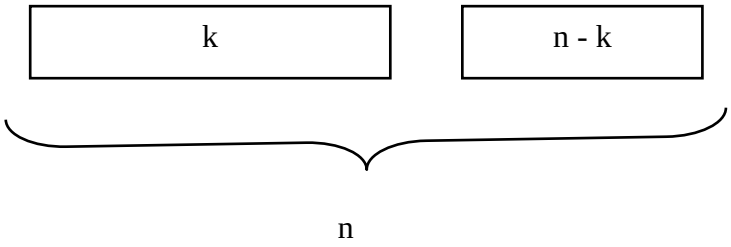


Рисунок 3.3 – Формат систематичного коду

Прості коди з перевіркою на парність (непарність). Це високошвидкісні коди з поганими корегуючими характеристиками [21]. До заданих k інформаційних біт дописується (k+1)-й біт так, щоб повне число одиниць в кодовому слові було парним *(parity)*. Цей додатковий біт називається бітом перевірки на парність. Такий код є (k + 1, k)-кодом або (n, n-1)-кодом. Кодова відстань цього коду рівна двом**,** тому ніякі помилки не можуть бути виправлені. Код з перевіркою на парність використовується для виявлення (але не виправлення) однієї помилки в кодовій комбінації.

Підрахунок кількості одиниць еквівалентний використанню арифметики за модулем 2. Припустимо, що інформаційна послідовність джерела має такий вигляд:

$$\vec{U}_i = (u_1, u_2, ..., u_k). \qquad (3.38)$$

Нехай

$$u_{k+1} = u_1 + u_2 + ... + u_k, \qquad (3.39)$$

де знак «+» - операція додавання за модулем 2. З (3.39) видно, що u_{k+1}=0, якщо сума в правій частині парна, і u_{k+1}=1, якщо вона непарна.

Припустимо, що замість послідовності $(u_1, u_2, ..., u_k)$ довжини k передається послідовність $(u_1, u_2, ..., u_k, u_{k+1})$ довжини k+1, що містить парне число символів «1» за рахунок передачі додаткового (надмірного) двійкового символу u_{k+1}. Якщо в процесі передачі такої послідовності

виникне помилка в одному з символів, то кількість одиниць стане непарною, тобто сума $u_1 + u_2 + ... + u_k + u_{k+1}$ буде рівною одиниці. Це і дозволяє виявляти виникнення одиничних помилок.

Приклад 3.2. Нехай передається повідомлення, що складається з чотирьох символів A_1, A_2, A_3, A_4, які подані такими кодовими комбінаціями $A_1 = 0\ 0$, $A_2 = 0\ 1$, $A_3 = 1\ 0$, $A_4 = 1\ 1$. Побудувати код з перевіркою на парність.

Розв'язування. Згідно з (3.38) сформуємо біт перевірки на парність для кожної кодової комбінації:

$$A_1 - 0 \oplus 0 = 0$$
$$A_2 - 0 \oplus 1 = 1$$
$$A_3 - 1 \oplus 0 = 1$$
$$A_4 - 1 \oplus 1 = 0$$

До заданих k = 2 інформаційних біт дописується (k+1)-й додатковий біт перевірки на парність, в результаті отримаємо новий код:

$$A_1 = 0\ 0\ 0$$
$$A_2 = 0\ 1\ 1$$
$$A_3 = 1\ 0\ 1$$
$$A_4 = 1\ 1\ 0.$$

Визначимо кодову відстань d_{min} для даного коду. Для цього визначимо кількість одиниць в сумі за модулем 2 всіх пар кодових комбінацій:

000	000	000	011	011	101
+	+	+	+	+	+
011	101	110	101	110	110
———	———	———	———	———	———
011	101	110	110	101	011
$d_1 = 2$	$d_2 = 2$	$d_3 = 2$	$d_4 = 2$	$d_5 = 2$	$d_6 = 2$

Таким чином, $d_{min}=2$ і даний код може виявляти лише одну помилку в кодовій комбінації (блоці). Якщо виникне дві помилки в межах одного блока, то вони переведуть помилкову комбінацію в іншу дозволену комбінацію з парною кількістю одиниць.

Прості коди з повторенням. Це низькошвидкісні коди з високими корегуючими характеристиками [21]. Один інформаційний символ повторюється n раз, тобто це (n,1)-коди. Звичайно n вибирають непарним. Наприклад (5,1) - код формується так:

0 – 00000,
1 – 11111.

Для даного коду кодова відстань дорівнює n. З (3.7) кількість помилок, які може виправляти цей код рівна:

$$(n-1)/2. \qquad (3.40)$$

Коди Хеммінга. Кодами Хеммінга називають звичайно такі лінійні коди [4]:

- коди з відстанню $d_{min}=3$, які виправляють всі одиничні помилки;
- коди з відстанню $d_{min}=4$, які виправляють всі одиничні помилки і виявляють подвійні.

Розглянемо перший випадок. Нехай є m незалежних перевірок на парність. Незалежні перевірки - це такі перевірки, у яких ніяка сума одних перевірок не збігається з будь-якою іншою перевіркою, причому, додавання виконується за модулем 2. Наприклад, перевірки на парність в позиціях

1: 1, 2, 5, 7
2: 5, 7, 8, 9
3: 1, 2, 8, 9

є залежними, оскільки сума будь-яких двох рядків збігається з третім.

Синдром (розпізнавач), який отримують, якщо написати символ "0" для перевірок, що виконались, і символ "1" для кожної перевірки, що не виконалась, можна розглядати як число, яке містить m = n-k двійкових символів. Це число може розрізняти не більше 2^m подій. З (3.11) для випадку виправлення однієї помилки одержимо:

$$2^m \geq n+1. \qquad (3.41)$$

Необхідно мати синдром, який дозволяв би виявляти правильні повідомлення, а також розташування помилки в будь-якій з n позицій повідомлення. Ідея, яка лежить в основі кодів Хеммінга, полягає в вимозі, щоб синдром давав фактичне розташування помилки, а рівний "0" означав відсутність помилки. Перевірки на парність, при цьому, повинні бути такі: кожна позиція, для якої остання цифра її номера,

записаного в двійковому поданні, дорівнює "1", повинна входити в першу перевірку на парність. Аналогічно, в другу перевірку на парність повинні входити позиції, у яких друга справа цифра її номера, записаного в двійковому поданні, рівна "1" і т. д. Таким чином, в першу перевірку на парність повинні входити позиції 1, 3, 5, 7, 9, 11,..., в другу - 2, 3, 6, 7, 10,..., в третю - 4, 5, 6, 7, 12, 13, 14,..., в четверту - 8, 9, 10, 11, 12, 13, 14, 15 і т.д.

Розглянемо приклад побудови коду, що виправляє помилку для 4-х двійкових символів. Оскільки інформаційних символів k=4, то з нерівності (3.39) для n = k+m знайдемо:

$$2^m \geq 5 + m.$$

Звідки m = 3. Тобто, необхідно мати m=3 перевірки на парність і загальна кількість символів (інформаційних і перевірних) складе:

$$n = 4+3 = 7.$$

Позиції, що відповідають степені “2”, тобто 1, 2, 4 використовуються для перевірних символів, а позиції 3, 5, 6, 7 - інформаційні. Тоді повідомлення буде мати такий вигляд:

Позиція	1	2	3	4	5	6	7
Повідомлення	_	_	1	_	0	1	1.

Пропуски указують позиції, де повинні знаходитись перевірні символи. Після першої перевірки на парність (позиції 1,3,5,7) отримаємо перший перевірний символ рівним "0" (сума за модулем 2 символів, що знаходяться в позиціях 3, 5, 7):

Позиція	1	2	3	4	5	6	7
Повідомлення	0	_	1	_	0	1	1.

Після другої перевірки на парність (позиції 2,3,6,7) отримаємо другий символ рівним "1" (сума за модулем 2 символів, що знаходяться в позиціях 3, 6, 7):

Позиція	1	2	3	4	5	6	7
Повідомлення	0	1	1	_	0	1	1.

Після третьої перевірки на парність (позиції 4,5,6,7) отримаємо третій перевірний символ рівним "0" (сума за модулем 2 символів, що знаходяться в позиціях 5, 6, 7) і повністю закодоване повідомлення:

Позиція	1	2	3	4	5	6	7
Повідомлення	0	1	1	0	0	1	1.

Нехай при передачі повідомлення виникла помилка в третьому символі зліва. Тобто прийнято повідомлення:

Позиція	1	2	3	4	5	6	7
Повідомлення	0	1	0	0	0	1	1.

Тоді після першої перевірки на парність (сума за модулем 2 символів в позиціях 1,3,5,7) отримаємо "1". Друга перевірка (сума за модулем 2 символів в позиціях 2,3,6,7) також дасть "1". Третя перевірка (сума за модулем 2 символів в позиціях 4,5,6,7) дасть "0".

Розглянувши отриманий синдром як двійкове число, отримаємо 011, що відповідає десятковому числу "3". Тому необхідно змінити символ в третій позиції прийнятого повідомлення. Отримаємо:

Позиція	1	2	3	4	5	6	7
Повідомлення	0	1	1	0	0	1	1.

Що відповідає переданому повідомленню, тобто помилка виправлена.

Коди з відстанню $d_{min}=4$, які виявляють всі одиничні помилки і виправляють подвійні, отримають шляхом введення додаткової перевірки на парність всієї кодової комбінації і ще однієї позиції. Тоді подвійна помилка призведе до деякого ненульового синдрому, але додаткова перевірка на парність буде виконана. Це і дозволяє розпізнати подвійну помилку.

Загальні принципи побудови двійкового групового коду. Побудова коду, який може виправляти більше однієї помилки більш складна задача у порівнянні з кодами Хеммінга. Задача побудови кодів Хеммінга є окремим випадком загальної задачі побудови групового коду, що виправляє декілька помилок. Побудова групового коду включає такі етапи [2].

1. Визначити кількість перевірних символів згідно з (3.9) для заданої кратності помилок. Необхідно підкреслити, що часто перевірних символів необхідно більше цієї теоретичної межі, це пов’язано з складністю визначення розпізнавачів для випадку виправлення декількох помилок.

2. Скласти таблицю розпізнавачів (синдромів). Наприклад, для коду (7,4), який виправляє одну помилку (код Хеммінга) таблиця розпізнавачів наведена в табл. 3.1.

Таблиця 3.1 – Таблиця розпізнавачів для коду (7,4)

Вектор помилок	Розпізнавач (синдром)	Вектор помилок	Розпізнавач
0000001	001	0010000	101
0000010	010	0100000	110
0000100	011	1000000	111
0001000	100		

Тут вектор помилки указує розряд, в якому виникла помилка, а синдром є двійковим поданням номера цього розряду.

Розглянемо більш складний випадок виправлення одиничних і подвійних незалежних помилок. Як синдром для одиничних помилок в першому і другому розряді можна прийняти, як і раніше, комбінації 0..001 та 0...010. Однак як синдром одиничної помилки в третьому розряді комбінацію 0...011 брати не можна, оскільки така комбінація відповідає помилці одночасно в першому і другому розрядах, яку також потрібно виправляти. Тоді як синдром одиничної помилки в третьому розряді можна взяти 0...100.

Вектор помилки 0...0101 (помилка в першому і третьому розряді) можна розглядати як результат сумарної дії двох векторів 0...0100 та 0...0001 і відповідно до цього йому може бути поставлений у відповідність розпізнавач, який подає суму за модулем 2 розпізнавачів цих помилок, тобто

$$\begin{array}{l} 0...0001 \\ \oplus \\ 0...0100 \\ \_\_\_\_\_ \\ 0...0101. \end{array}$$

Аналогічно знайдемо, що розпізнавач для вектора помилки 0...0110 є комбінацією 0...0110.

Визначаючи розпізнавач для одичної помилки в четвертому розряді помічаємо, що невикористана трирозрядна комбінація 111. Для векторів подвійних помилок в четвертому і молодших розрядах (0...01001, 0...01010, 0...01100) визначимо розпізнавачі:

$$\begin{array}{|l|} 0...0111 \\ \oplus \\ 0...0001 \\ \hline 0...0110 \end{array} \quad \begin{array}{|l|} 0...0111 \\ \oplus \\ 0...0010 \\ \hline 0...0101 \end{array} \quad \begin{array}{|l|} 0...0111 \\ \oplus \\ 0...0100 \\ \hline 0...0011 \end{array}.$$

Однак, ці комбінації уже використовувались для векторів помилок 0...0110, 0...0101, 0...0011 і комбінація 0...0111 не може бути використана як розпізнавач для помилки в четвертому розряді, тому як розпізнавач помилки в четвертому розряді необхідно взяти комбінацію 1000. Тобто, вибираючи розпізнавач для помилки в “i”-у розряді з числом розрядів меншим “i”, необхідно перевіряти чи для інших векторів помилок з “1” в “i”-у розряді розпізнавачі будуть відрізнятись від уже отриманих.

За допомогою ЕОМ отримані розпізнавачі для кодів даного типу включно до 29-го розряду. Розпізнавачі одиничних помилок в перших 15 розрядах наведені в табл. 3.2. Розпізнавачі для подвійних помилок можна отримати з даної таблиці як суми за модулем 2 розпізнавачів відповідних одиничних помилок. Подібним способом можна визначити таблиці розпізнавачів і для інших типів помилок: потрійних незалежних помилок, пакетів помилок і т.п.

Таблиця 3.2 – Таблиця розпізнавачів для одиничних помилок

Номер розряда	Розпізнавач (синдром)	Номер розряда	Розпізнавач (синдром)	Номер розряда	Розпізнавач (синдром)
1	0000 0001	6	0001 0000	11	0110 1010
2	0000 0010	7	0010 0000	12	1000 0000
3	0000 0100	8	0011 0011	13	1001 0110
4	0000 1000	9	0100 0000	14	1011 0101
5	0000 1111	10	0101 0101	15	1101 1011

3. Визначити перевірні рівності. Нехай після першої перевірки на парність для молодшого розряду розпізнавача отримана одиниця (табл. 3.1). Це може бути наслідком помилки в одному з розрядів, розпізнавачі яких мають одиницю в молодшому розряді. Тому перша перевірна рівність повинна включати символи 1, 3, 5, 7-го розрядів:

$$a_1 \oplus a_3 \oplus a_5 \oplus a_7 = 0. \qquad (3.42)$$

Одиниця в другому розряді розпізнавача може бути наслідком помилки в розрядах, розпізнавачі яких мають одиниці в другому розряді. Звідки отримаємо другу перевірну рівність:

$$a_2 \oplus a_3 \oplus a_6 \oplus a_7 = 0. \qquad (3.43)$$

Аналогічно знаходимо третю перевірну рівність:

$$a_4 \oplus a_5 \oplus a_6 \oplus a_7 = 0. \qquad (3.44)$$

Оскільки ми розглядаємо код (7, 4), який має три перевірних символи, то необхідно вибрати номери розрядів, де вони будуть знаходитись. Для однозначного визначення значення цих символів необхідно, щоб кожний з них входив лише в одну перевірну рівність. Для нашого випадку це будуть перший, другий і четвертий розряди. Виходячи з цього визначимо перевірні символи:

$$\left.\begin{aligned} a_1 &= a_3 \oplus a_5 \oplus a_7 \\ a_2 &= a_3 \oplus a_6 \oplus a_7 \\ a_4 &= a_5 \oplus a_6 \oplus a_7 \end{aligned}\right\}. \qquad (3.45)$$

Побудова кодів, що виправляють дві або більше помилок для достатньо великих блоків, вкрай важка без моделювання на ЕОМ.

Приклад 3.3. Використовуючи табл. 3.2 знайти правила побудови коду (8,2), який виправляє всі одиничні та подвійні помилки.

Розв'язування. З табл. 3.2 видно, що при восьми розрядах (n=8), кількість перевірних символів шість і відповідні перевірні рівності такі:

$$\begin{aligned} a_1 \oplus a_5 \oplus a_8 &= 0, \\ a_2 \oplus a_5 \oplus a_8 &= 0, \\ a_3 \oplus a_5 &= 0, \\ a_4 \oplus a_5 &= 0, \\ a_6 \oplus a_8 &= 0, \\ a_7 \oplus a_8 &= 0. \end{aligned}$$

Звідки визначимо перевірні символи:

$$\begin{aligned} a_1 &= a_5 \oplus a_8, \\ a_2 &= a_5 \oplus a_8, \\ a_3 &= a_5, \\ a_4 &= a_5, \\ a_6 &= a_8, \\ a_7 &= a_8. \end{aligned}$$

Для даного коду $d_{min}=5$ і він згідно з виразами (3.6, 3.7) може виправляти одиничні і подвійні помилки і виявляти помилки кратності від 1 до 4.

Циклічні коди. Циклічні коди є різновидністю поліноміальних кодів [25]. В таких кодах вважається, що елементи a_1, a_2, …a_{n-1} деякого кодового слова є коефіцієнтами полінома від x:

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1}. \quad (3.46)$$

Використовуючи це позначення можна визначити поліноміальний код як множину всіх многочленів степені не більше чим n-1, які містять як множник деякий фіксований многочлен g(x), який називається породжувальним многочленом. Тоді процес кодування можна подати як результат множення полінома m(x), що являє собою інформаційну послідовність на породжувальний многочлен g(x), а декодування як результат ділення на цей поліном.

При деяких значеннях n поліноміальні ходи характеризуються властивістю циклічності. Це означає, що циклічна перестановка кодового слова знову приводить до кодового слова даного коду (див. п. 3.6).

Циклічні коди привабливі з двох причин:

- по-перше, операції кодування і обчислення синдрому для них виконуються дуже просто з використанням регістрів зсуву;
- по-друге, цим кодам властива алгебраїчна структура, і можна знайти простіші і ефективніші способи їх декодування.

Двійкові циклічні коди утворюють лінійні векторні простори над полем Галуа GF(2). На практиці широко використовуються циклічні коди з компонентами з розширених полів Галуа $GF(2^m)$. Такими кодами є коди Боуза-Чоудхурі-Хоквінгема (БЧХ) і коди Ріда-Соломона (РС) [19, 23-24]. Коди РС, зокрема, використовуються в програвачах компакт-дисків [19].

Основні властивості циклічних (n,k)-кодів такі [25].

1. У циклічному (n,k) -коді кожен ненульовий поліном повинен мати степінь, принаймні (n-k), але не більш n-1.
2. Існує тільки один кодовий поліном g(x) степені (n-k)

$$g(x)=1+g_1x+g_2x^2+...+g_{n-k}x^{n-k}, \quad (3.47)$$

   який називається породжувальним поліномом коду.
3. Кожен кодовий поліном u(x) є кратним g(x), тобто

$$u(x)=m(x)\cdot g(x). \quad (3.48)$$

4. Ще однією важливою властивістю циклічного (n,k)-кода є те, що його породжувальний поліном ділить без залишку двочлен x^n+1,

тобто $x^n+1=g(x)h(x) + 0$. Поліном h(x) — частка від ділення x^n+1 на g(x) - називається перевірним поліномом. Оскільки h(x) однозначно пов'язаний з g(x), він також визначає код. Отже, за допомогою перевірного полінома h(x) теж можна проводити кодування.

Розглянемо процес побудови циклічного коду в полі GF (2). Якщо просто помножити інформаційний поліном на перевірний згідно з властивістю 3, то отриманий код хоча і придатний для кодування, але не є систематичним.

Для того щоб отримати систематичний циклічний код помножимо поліном, що відповідає інформаційній послідовності $m=(m_0, m_1,\ldots,m_{k-1})$, на x^{n-k}. В результаті отримаємо поліном n-1 степені або менше:

$$x^{n-k}m(x) = m_0x^{n-k} + m_1x^{n-k+1} + \ldots + m_{k-1}x^{n-1}. \qquad (3.49)$$

Скориставшись теоремою про ділення поліномів, можна записати:

$$x^{n-k}m(x) = q(x)g(x) + p(x), \qquad (3.50)$$

де *q(x)* і *p(x)* - частка і залишок від ділення полінома на породжувальний поліном *g(x).*

Оскільки степінь *g(x)* рівний *(n-k),* то степінь *p(x)* повинен бути *(n-k-1)* або менше, а сам поліном *p(x)* матиме вигляд:

$$p(x) = p_0 + p_1x + \ldots + p_{n-k-1}x^{n-k-1}. \qquad (3.51)$$

З урахуванням правил арифметики в *GF(2)* вираз (3.50) можна переписати таким чином:

$$p(x) + x^{n-k}m(x) = q(x)g(x). \qquad (3.52)$$

Звідки видно, що поліном $p(x) + x^{n-k}m(x)$ є кратним *g(x)* і має степінь *n-1* або менший. Отже, поліном $p(x) + x^{n-k}m(x)$ - це кодовий поліном, відповідний кодованій інформаційній послідовності *m(x).*

Розкривши останній вираз, отримаємо:

$$\begin{aligned} p(x) + x^{n-k}m(x) &= p_0 + p_1x + \ldots + p_{n-k-1}x^{n-k-1} + \\ &+ m_0x^{n-k} + m_1x^{n-k+1} + \ldots + m_{k-1}x^{n-1}. \end{aligned} \qquad (3.53)$$

Що відповідає кодовому слову

$$U = (p_0, p_1, ... p_{n-k-1}, m_0, m_1, ... m_{k-1}), \qquad (3.54)$$

де $p_0, p_1, ... p_{n-k-1}$ - перевірні символи;

$m_0, m_1, ... m_{k-1}$ - інформаційні символи.

Таким чином, кодове слово складається з незмінної інформаційної частини *m,* перед якою розташовано *(n-k)* перевірних символів. Перевірні символи є коефіцієнтами полінома *p(x),* тобто залишку від ділення $p(x) + x^{n-k} m(x)$ на породжувальний поліном *g(x).*

Нагадаємо, що множенню деякого двійкового полінома на x^{n-k} відповідає зсуву двійкової послідовності $m = (m_0, m_1, ... m_{k-1})$ на *n-k* розрядів вправо.

Отже, систематичний циклічний (n,k) код k-розрядної інформаційної послідовності $m = (m_0, m_1, ... m_{k-1})$ отримують таким чином:

- інформаційну послідовність m множать на x^{n-k}, тобто зсувають вправо на n-k розрядів;

- поліном отриманої послідовності ділять на породжувальний поліном коду g(x);

- отриманий залишок від ділення $x^{n-k} m(x)$ на g(x) додають до $x^{n-k} m(x)$, тобто записують в молодших n-k розрядах коду.

Алгоритм кодування, оснований на діленні поліномів, можна реалізувати, використовуючи схему ділення. Вона є регістром зсуву, в якому ланцюги зворотного зв'язку замкнуті відповідно до коефіцієнтів породжувального полінома g(x) (рис. 3.4). Кодування згідно з даною схемою виконується таким чином:

- k символів інформаційної послідовності m через перемикач П, що знаходиться у верхньому положенні, один за одним передаються в канал і одночасно з цим через відкриту схему І записуються в регістр перевірних символів, в якому завдяки наявності ланцюгів зворотного зв'язку $g_0, g_1, ... g_{n-k-1}$ формується залишок від ділення $x^{n-k} m(x)$ на g(x) - перевірні символи;

- починаючи з k+1-го такту перемикач переводиться в нижнє положення, і з регістра зсуву виводяться (n-k) перевірних символів: ланцюг зворотного зв'язку при цьому розімкнений (схема І - закрита).

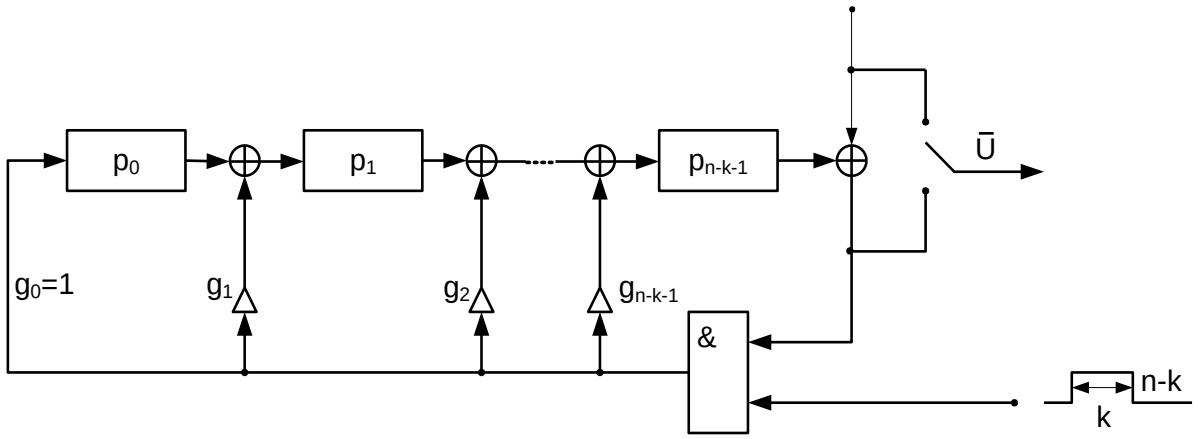


Рисунок 3.4 – Узагальнена схема циклічного систематичного кодера

Приклад 3.4. З використанням коду, що задається породжувальним поліномом $g(x) = 1 + x + x^3$, закодувати послідовність m = (0111).

Розв'язування. Послідовності m =(0111) відповідає поліном $m(x) = x + x^2 + x^3$. Помножимо $m(x)$ на x^{n-k}:

$$x^{n-k} m(x) = m(x)x^3 = x^3(x + x^2 + x^3) = x^4 + x^5 + x^6.$$

Розділимо $m(x)x^{n-k}$ на породжувальний поліном $g(x)$:

$$x^6 + x^5 + x^4 \qquad \left| \underline{x^3 + x + 1} \right.$$

$$\underline{x^6 + 0 + x^4 + x^3} \qquad x^3 + x^2 = q(x)$$

$$x^5 + 0 + x^3$$

$$\underline{x^5 + 0 + x^3 + x^2}$$

$$x^2 = p(x)$$

Таким чином, кодовий поліном для інформаційної послідовності m= (0111), матиме такий вигляд:

$$u(x) = 0 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 0 \cdot x^3 + 1 \cdot x^4 + 1 \cdot x^5 + 1 \cdot x^6,$$

а відповідне кодове слово U = (0010111).

Код, з породжувальним поліномом $g(x) = 1 + x + x^3$, який розглянуто в прикладі, називається циклічним (7,4) кодом Хеммінга (цей код має властивість циклічності).

Схема кодування для цього коду наведена на рис. 3.5. У цій схемі, на відміну від узагальненої схеми кодера на рис. 3.4, відсутні елементи в ланцюгах, де значення коефіцієнтів зворотного зв'язку g_i рівні нулю, а там де ці коефіцієнти рівні одиниці ланцюг просто замкнутий.

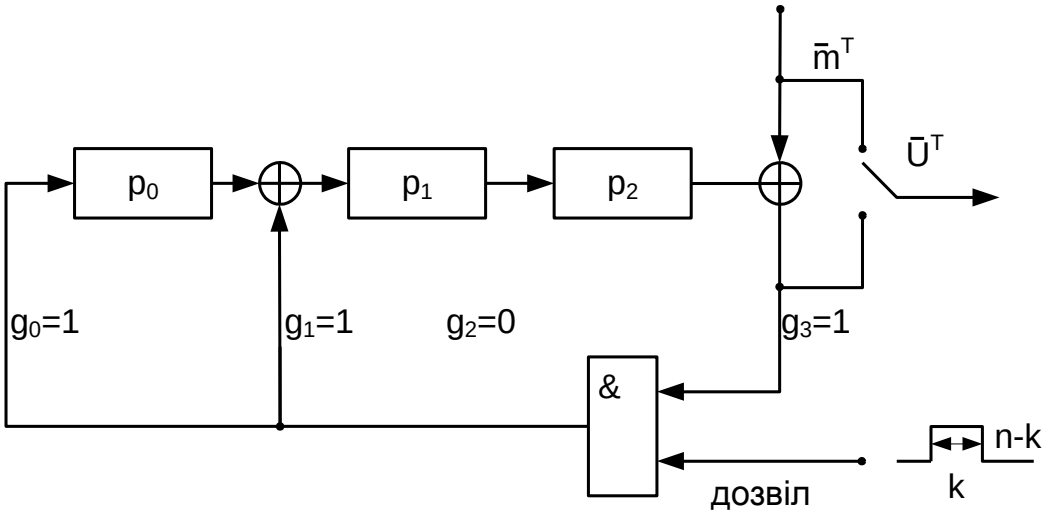


Рисунок 3.5 – Схема циклічного систематичного кодера для коду (7,4)

Декодування циклічних кодів може виконуватись як із застосуванням алгебраїчних методів, так і неалгебраїчних. В основі алгебраїчних методів лежить розв'язок систем рівнянь, що задають значення і розташування помилок. При неалгебраїчних методах та ж мета досягається за рахунок простих структурних понять теорії кодування, що спрощують знаходження помилок. Прикладом такого підходу може бути алгоритм Меггіта [25]. Однак із розвитком елементної бази можливе зниження інтересу до таких спеціальних методів. З урахуванням цього розглянемо синдромне декодування циклічних кодів, яке відноситься до алгебраїчних методів.

Нехай *u(x)* і *r(x)* - поліноми, відповідні переданому кодовому слову і прийнятій послідовності. Розділивши *r(x)* на *g(x),* отримаємо:

$$r(x)=q(x)g(x)+s(x), \qquad (3.55)$$

де - *q(x)* — частка від ділення;

s(x) — залишок від ділення.

*Я*кщо *r(x)* є кодовим поліномом, то він ділиться на *g(x)* без залишку, тобто *s(x)*= 0. Отже, s(x) <>0 є умовою наявності помилки в прийнятій послідовності, тобто синдромом прийнятої послідовності, який однозначно визначає як наявність, так і місце помилки. Синдром s(x) має в загальному випадку вигляд:

$$S(x)=s_0\cdot x^0+s_1\cdot x^1+s_2\cdot x^2+...+s_{n-k-1}\cdot x^{n-k-1}. \qquad (3.56)$$

Очевидно, що схема обчислення синдрому є схемою ділення, подібною до схем кодування рис. 3.4 - 3.5. За наявності в прийнятій

послідовності *r* хоч би однієї помилки вектор синдрому *S* матиме, принаймні, один ненульовий елемент, при цьому факт наявності помилки легко виявити, об'єднавши за *АБО* виходи всіх комірок регістра синдрому.

3.8 Згортні коди

Лінійні деревовидні коди називають згортними кодами [19, 20-22]. Вони є найпростішими з точки зору реалізації і тому знайшли найбільш широке застосування. Згортні коди відіграють провідну роль в сучасних системах зв'язку. Це в першу чергу відноситься до цифрового радіомовлення і до мобільного зв'язку мережі GSM. Останніми роками згортні коди отримали подальший розвиток у зв'язку з відкриттям турбо-кодів. Досить сказати, що використання турбокодів в сучасних системах передачі даних дозволило досягти швидкостей передачі інформації близьких до пропускної спроможності каналу. Найважливішими відмінностями згортних кодів від блокових є такі [19].

1. Згортні коди дозволяють проводити кодування і декодування потоків даних безперервно в часі.
2. Згортні коди не потребують блокової синхронізації.
3. Застосування згортних кодів дозволяє досягти дуже високої надійності передачі інформації.
4. Хороші згортні коди можуть бути знайдені шляхом моделювання.

Дані коди називаються згортними, оскільки виконують згортку послідовності інформаційних символів з імпульсною характеристикою кодера. Тут кодер розглядається як цифровий фільтр і імпульсна характеристика кодера визначається як реакція кодера на одиничний імпульс (1000...).

Приклад 3.5. Визначити згортку двійкових послідовностей g(m)=(1,0,1,1) та k(n)=(1,0,1), де g(m) – імпульсний відгук кодера, а k(n) – інформаційна послідовність.

Розв'язування. Згортка задається таким виразом:

$$u(n) = k(n) \otimes g(n) = \sum_{m=0}^{m} g(m)k(n-m).$$

Оскільки g(m) містить чотири двійкові символи, то m = 0,1,2,3, а вираз для u(n) буде таким:

$$u(n) = k(n) \otimes g(n) = \sum_{m=0}^{3} g(m)k(n-m).$$

Знайдемо елементи u(n):

$$u(0) = g(0)\cdot k(0) \oplus g(1)\cdot k(0-1) \oplus g(2)\cdot k(0-2) \oplus g(3)\cdot k(0-3) = 1\cdot 1 \oplus 0\cdot 0 \oplus 1\cdot 0 \oplus 1\cdot 0 = 1;$$
$$u(1) = g(0)\cdot k(1) \oplus g(1)\cdot k(0) \oplus g(2)\cdot k(-1) \oplus g(3)\cdot k(-2) = 1\cdot 0 \oplus 0\cdot 1 \oplus 1\cdot 0 \oplus 1\cdot 0 = 0;$$
$$u(2) = g(0)\cdot k(2) \oplus g(1)\cdot k(1) \oplus g(2)\cdot k(0) \oplus g(3)\cdot k(-1) = 1\cdot 1 \oplus 0\cdot 0 \oplus 1\cdot 1 \oplus 1\cdot 0 = 0;$$
$$u(3) = g(0)\cdot k(3) \oplus g(1)\cdot k(2) \oplus g(2)\cdot k(1) \oplus g(3)\cdot k(0) = 1\cdot 0 \oplus 0\cdot 1 \oplus 1\cdot 0 \oplus 1\cdot 1 = 1;$$
$$u(4) = g(0)\cdot k(4) \oplus g(1)\cdot k(3) \oplus g(2)\cdot k(2) \oplus g(3)\cdot k(1) = 1\cdot 0 \oplus 0\cdot 0 \oplus 1\cdot 1 \oplus 1\cdot 0 = 1;$$
$$u(5) = g(0)\cdot k(5) \oplus g(1)\cdot k(4) \oplus g(2)\cdot k(3) \oplus g(3)\cdot k(2) = 1\cdot 0 \oplus 0\cdot 0 \oplus 1\cdot 0 \oplus 1\cdot 1 = 1.$$

Кодер для згортного коду є пристроєм з пам'яттю, в який поступають набори з k_0 вхідних символів, а на виході з'являються набори з n_0 вихідних символів. Кожен набір n_0 вихідних символів залежить від поточного вхідного набору і від *K-1* попередніх вхідних наборів. Параметр *K* називається *конструктивною довжиною* згортного коду, а величина $n_A=Kn_0$ - *довжиною кодового обмеження.* Деревовидні коди характеризуються також швидкістю $r=k_0/n_0$ і *вільною відстанню* d_{free}, сенс якої буде пояснюватися далі.

Для побудови кодера згортного коду необхідно k_0 регістрів зсуву, зв'язки між елементами яких визначаються набором породжувальних поліномів $g_{i,j}(x)$, де i=0, 1 ... k_0-1- номер вхідного потоку, а j = 0, 1 ..., n_0-1 - номер вихідного потоку. Оскільки на практиці частіше використовуються коди з єдиним вхідним потоком (k_0=1), індекс "i" зазвичай опускається, але в будь-якому випадку на відміну від блокових кодів, які описуються єдиним породжувальним поліномом, згортний код потребує для свого опису декілька породжувальних поліномів. Ще однією характеристикою згортного коду є його конструктивна довжина K, що впливає на складність його декодування і рівна довжині найдовшого регістра зсуву. Для прикладу на рис. 3.6 поданий кодер несистематичного згортного коду, заданий породжувальними поліномами:

$$G_0(x)=1+x^2;$$
$$G_1(x)= 1 + x + x^2 .$$

Даний код характеризується такими параметрами: кількістю інформаційних гілок k_0=1, кількістю перевірних гілок n_0=2, кодовою швидкістю r=1/2, довжиною кодового обмеження n_A=6, конструктивною довжиною коду K=3.

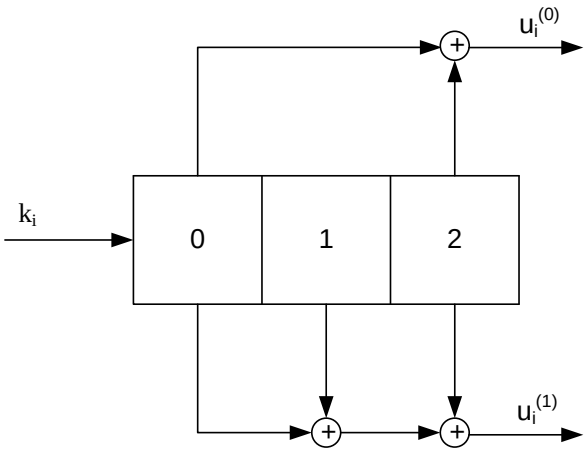


Рисунок 3.6 - Кодер згортного несистематичного коду

Так само як і блокові, згортні коди можна подати у систематичній формі, окремим випадком якої є рекурсивні систематичні згортні (Recursive Systematic Convolutional - RSC) коди. Характерною особливістю кодерів даних кодів є наявність зворотного зв'язку. Як приклад на рис. 3.7 поданий RSC кодер, заданий поліномом зворотного зв'язку (feedback polynomial) $G_0(x)= 1 + x + x^2$ і вихідним поліномом (feedforward polynomial) $G_1(x)= 1 + x^2$.

Можливі різні способи опису згортних кодів, наприклад з використанням породжувальної матриці. Однак через нескінченність послідовності, що кодується, і породжувальна матриця G буде мати нескінченні розміри. Більш зручним способом задання згортних кодів є їх задання за допомогою імпульсної характеристики або відповідних їй породжувальних поліномів (поліноміальне подання).

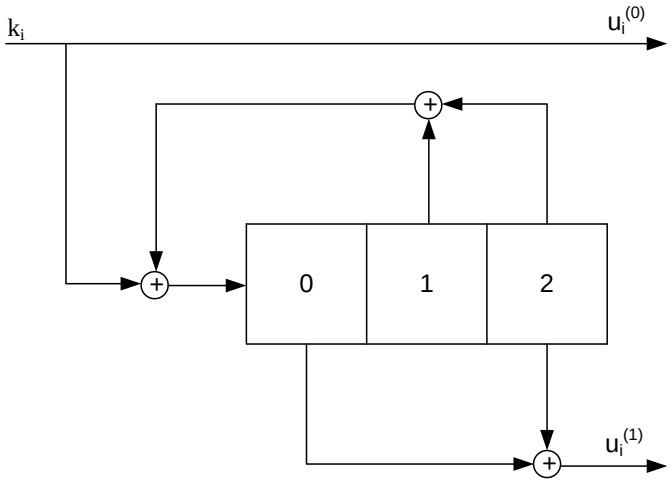


Рисунок 3.7 - Кодер згортного систематичного коду

Наприклад, для кодера на рис. 3.6 імпульсна характеристика така:

$$g=(11\ 10\ 11\ 00\ 00\ \ldots).$$

А відповідні породжувальні поліноми такі:

$$\left.\begin{aligned} G^1(x) &= 1 \oplus x \oplus x^2 \\ G^2(x) &= 1 \oplus x^2 \end{aligned}\right\}. \qquad (3.57)$$

Якщо записати коефіцієнти поліномів в двійковій формі, отримаємо:

$$G^1(x) = 111$$

$$G^2(x) = 101.$$

Для довгих кодів використовують восьмеричну форму запису:

$$G^1(x) = 7,$$

$$G^2(x) = 5,$$

або G=(7,5). Процес кодування може бути поданий як результат множення многочлена вхідної послідовності на породжувальний многочлен кода:

$$u_i(x) = k(x) \cdot G_i(x). \qquad (3.58)$$

Приклад 3.6. Закодувати інформаційну послідовність k=(1011100…) з використанням імпульсної характеристики g=(11 10 11 00 00 …).

Розв'язування. Кодування виконується шляхом зсуву імпульсної характеристики на два біти вправо для кожного інформаційного символу із заповненням вільних бітів нулями і додаванням за модулем 2 результатів зсуву, відповідних одиничним символам вхідної послідовності:

$$\begin{array}{l} U = 11\ \ 10\ \ 11\ \ 00\ \ 00\ \ 00 \\ \qquad\quad 11\ \ 10\ \ 11\ \ 00\ \ 00\ \ 00 \\ \qquad\qquad 11\ \ 10\ \ 11\ \ 00\ \ 00\ \ 00 \\ \qquad\qquad\qquad 11\ \ 10\ \ 11\ \ 00\ \ 00\ \ 00 \\ \hline \qquad 11\ \ 10\ \ 00\ \ 01\ \ 10\ \ 01\ \ 11\ \ 00\ \ 00\ \ 00\ \ \ldots \end{array}$$

Приклад 3.7. Закодувати інформаційну послідовність k=(1011100…) з використанням кодера G=(7,5).

Розв'язування. Процес кодування може бути поданий як результат множення многочлена вхідної послідовності на породжувальні многочлени коду. Для даного кодера породжувальні многочлени такі:

$$G^1(x) = 1 \oplus x \oplus x^2$$

$$G^2(x) = 1 \oplus x^2.$$

Многочлен вхідної послідовності такий: $k(x) = 1 \oplus x^2 \oplus x^3 \oplus x^4$.

Тоді отримаємо:

$u_1(x) = k(x) \cdot G^1(x) = (1 \oplus x^2 \oplus x^3 \oplus x^4) \cdot (1 \oplus x \oplus x^2) = 1 \oplus x \oplus x^4 \oplus x^6$;

$u_2(x) = k(x) \cdot G^1(x) = (1 \oplus x^2 \oplus x^3 \oplus x^4) \cdot (1 \oplus x^2) = 1 \oplus x^3 \oplus x^5 \oplus x^6$.

Що відповідає кодовим послідовностям:

$U_1 = 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \ ...$

$U_2 = 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \ ...$

$U = 11 \ 10 \ 00 \ 01 \ 10 \ 01 \ 11 \ 00 \ 00...$

Тобто, аналогічно попередньому прикладу.

Згортний кодер як автомат з кінцевим числом станів може бути описаний діаграмою станів. Такий підхід є ключовим і веде до глибшого розуміння властивостей згортних кодів. Більш того, він сприяє розробці ефективних алгоритмів кодування і декодування.

Внутрішній стан кодера визначають K-1 розрядів регістра зсуву, починаючи від входу кодера. Для кодера G=(7,5) з K=3 діаграма станів наведена на рис. 3.8. Діаграма являє собою направлений граф, який описує всі можливі переходи кодера з одного стану в інший, а також містить значення виходів кодера, які супроводжують ці переходи. В кружках показані чотири можливих стани кодера: S_1S_2=00, 10, 11, 01. Суцільними лініями позначені переходи, що відповідають надходженню на вхід кодера інформаційного символу “0”, а пунктирними - символу “1”.

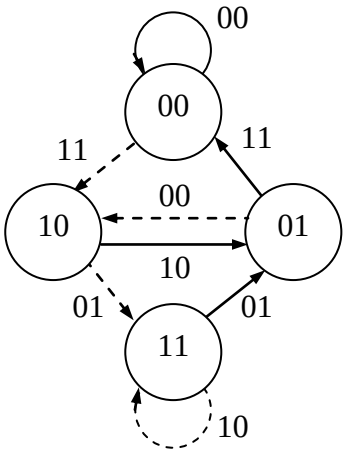


Рисунок 3.8 - Діаграма станів кодера

Розгорткою діаграми станів в часі є гратчаста діаграма (рис. 3.9). На гратці стани показані вузлами, а переходи лініями, що їх з'єднують. Після кожного переходу з одного стану в інший виконується зсув на один крок вправо. Ця діаграма дає наглядне уявлення всіх дозволених шляхів по яких може просуватись кодер при кодуванні. Кожній інформаційній послідовності на вході кодера відповідає єдиний шлях на гратці.

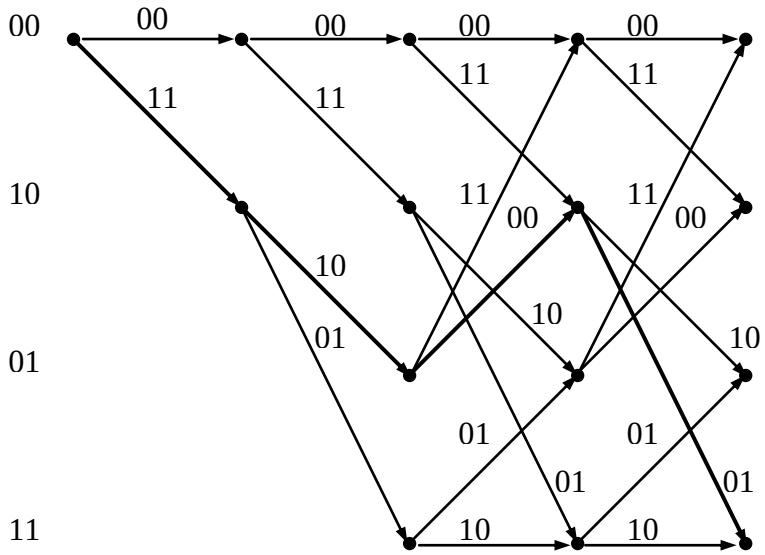


Рисунок 3.9 – Гратчаста діаграма згортного коду

Кодування згортними кодами з використанням гратчастої діаграми являє собою доволі просту процедуру: чергові символи вхідної послідовності визначають напрям руху з вузлів гратки – "0" – по верхньому ребру, "1" – по нижньому ребру. Для вхідної послідовності k=(1011…) шлях по гратчастій діаграмі (жирна лінія) 11 10 00 01… відповідає початку кодової послідовності U для прикладів 3.6 – 3.7.

Ще одним способом подання згортних кодів є кодове дерево. Однак його недоліком є експоненціальне зростання кількості гілок із зростанням довжини вхідної послідовності, тоді як гратчаста діаграма, починаючи з третього кроку, має постійну ширину.

Використовуючи гратчасте подання визначимо поняття вільної відстані d_{free} як мінімальної відстані між двома різними шляхами по кодовій гратці, які починаються і закінчуються в одному стані. Наприклад, з рис. 3.9 мінімальна вільна відстань для даного коду дорівнює "5" – загальна вага шляху $0\rightarrow1\rightarrow2\rightarrow0$, заданого вершинами на діаграмі.

При аналізі корегуючої здатності коду d_{free} відіграє таку ж роль, як мінімальна кодова відстань для блокових кодів. Згортний код вважається оптимальним, якщо при заданій швидкості коду і пам'яті він має максимальне значення d_{free}.

Задача знаходження хорошого згортного коду є задачею пошуку множини взаємно простих породжувальних многочленів. Тобто, породжувальні многочлени згортного коду повинні задовольняти таку рівність:

$$\gcd(g_1(x), g_2(x), ..., g_{n_0}(x)) = x^n, \qquad (3.59)$$

де gcd –найбільший спільний дільник (greatest common divisor);

$g_1(x), g_2(x), ..., g_{n_0}(x)$ - породжуючі многочлени коду, $x^n = 1$.

Коди, які задовольняють дану умову, називаються некатастрофічними, інакше коди називаються катастрофічними. В катастрофічних кодах кінцева кількість помилок в кодових символах викликає нескінченну кількість бітових помилок в декодованих даних.

Знайти множину взаємно простих многочленів неважко, однак знайти множину, яка добре виправляє помилки, важко. Оптимальні коди знаходять шляхом комп'ютерного моделювання. У літературі наведені численні таблиці породжувальних многочленів оптимальних згортних кодів [19].

Декодування згортних кодів. Декодування згортних кодів можна виконувати алгебраїчними та імовірнісними методами [19, 25 -27].

Алгебраїчні методи (синдромне та мажоритарне (порогове) декодування) грунтуються на використанні алгебраїчних властивостей кодових послідовностей. В деяких випадках такі методи приводять до простих реалізацій кодеків, однак вони не є оптимальними, оскільки алгебраїчні процедури декодування призначені для виправлення конкретних конфігурацій помилок в каналі.

Імовірнісні методи декодування значно ближчі до оптимального прийому в цілому, оскільки декодер оперує з величинами пропорційними імовірностям, оцінює і порівнює імовірності різних гіпотез і на цій основі виносить рішення про передані символи.

В системах зв'язку із згортними кодами використовують в основному два алгоритми декодування:

- алгоритм Вітербі та його модифікації (імовірнісний метод);

- пороговий алгоритм (алгебраїчний метод).

Алгоритм Вітербі є типовим алгоритмом декодування з використанням імовірнісних характеристик прийнятих символів і є алгоритмом максимальної правдоподібності. Використовується при декодуванні коротких згортних кодів.

Розглянемо більш детально декодування згідно з алгоритмом Вітербі.

Алгоритм Вітербі. Для двійкового симетричного каналу без пам'яті (канал, в якому імовірності передачі "0" і "1", а також імовірності помилок вигляду 0-->1 і 1-->0 однакові, помилки в символах коду незалежні) декодер максимальної правдоподібності зводиться до декодера мінімальної хеммінгової відстані. Обчислюється відстань Хеммінга між прийнятою послідовністю X і всіма можливими кодовими векторами U_i і приймається рішення на користь того вектора, який найближчий до прийнятого.

Для декодування використовується та ж сама гратчаста діаграма, що і при кодуванні. Періодична структура гратчастої діаграми суттєво спрощує порівняння і вибір шляху по гратчастій діаграмі. Кількість станів на гратці обмежена і два навгад вибраних шляхи мають, як правило, спільні стани. Відрізки шляхів, що входять в ці стани, необхідно порівняти і вибрати шлях з найменшою метрикою. Такий шлях називається вижившим. Для подальшого пояснення введемо такі поняття [27]:

- метрика гілки (МГ) – дорівнює відстані Хеммінга між набором символів $x^{(1)}x^{(2)}$ на вході декодера і набором символів $a^{(1)}a^{(2)}$, що відповідають даній гілці на гратчастій діаграмі;

- метрика шляху (МШ) - сума метрик гілок, що утворюють цей шлях на гратчастій діаграмі;

- метрика стану (МС) – дорівнює метриці шляху, що закінчується в даному стані.

Відповідно до алгоритму Вітербі на кожному кроці декодування в кожному стані гратчастої діаграми виконуються такі однотипні операції:

- додавання метрик попередніх станів з метриками відповідних гілок;

- порівняння метрик шляхів, що входять в кожний стан, і вибір шляху з найменшою метрикою (вижившій шлях), величина якої використовується як метрика стану на наступному кроці декодування.

Якщо метрики шляхів однакові, то вибір одного з двох шляхів виконується випадковим чином.

При відслідковуванні на велику глибину гратки, вижившi шляхи, як правило, зливаються. В цей момент згідно з алгоритмом Вітербі і приймається рішення про передану послідовність. Глибина, на якій зливаються шляхи, є величиною випадковою і залежить від кратності і імовірності помилок в каналі. На практиці не чекають злиття шляхів, а встановлюють фіксовану глибину декодування, яку звичайно вибирають в діапазоні $n_A<b\le n_A+q$, q – кратність помилок, що виправляються даним кодом. До того ж пам'ять декодера не може бути нескінченною величиною. Тому алгоритм Вітербі не є строго оптимальним.

Розглянемо приклад з використанням гратчастої діаграми на рис. 3.9 для кодера з породжувальними поліномами G=(7, 5). Нехай передана кодова послідовність U = (00000…), а прийнята r = (10000…), тобто в першому кадрі кодового слова виникла помилка. Процедуру декодування ілюструє рис. 3.10.

В початковий момент часу декодер знаходиться в стані 00 і початкова метрика цього стану МС (00) = 0.

Якщо на вхід декодера поступила пара символів r=10, то метрики гілок 00 і 11, що виходять з даного стану, будуть МГ(00)=1 і МГ(11)=1. Оскільки інших гілок, які виходять з стану 00 в стан 00 і 10, немає і метрика попереднього стану МС (00) = 0, то МС (00) =1 і МС (10) =1. На наступному кроці декодування, коли на вхід декодера поступає пара символів 00, тобто прийнята послідовність рівна r=10 00, метрики станів визначаються як сума метрик гілок, що входять в ці стани і метрик попередніх станів:

МС (00) = 1 + 0 = 1;
МС (10) = 2 + 1 = 3;
МС (01) = 1 + 1 = 2;
МС (00) = 1 + 1 = 2.

На цьому процес розвитку діаграми закінчується. Починаючи з рівня три є два шляхи, що ведуть у кожну вершину відповідного рівня. Порівнюючи метрики шляхів, що входять в кожний стан, вибираємо шлях з найменшою метрикою. Шляхи, що вижили при прийомі r=10 00 00, r=10 00 00 00, r=10 00 00 00 00 наведено на рис. 3.10. Причому після прийому r=10 00 00 00 шляхи, що входили в стан 10, мали однакову

метрику 3, а після прийому r=10 00 00 00 00 метрики шляхів, що входили в стан 10, 01 і 11 також мали однакову метрику (3, 4, 4, відповідно). В цьому випадку завжди вибирався верхній шлях на гратчастій діаграмі.

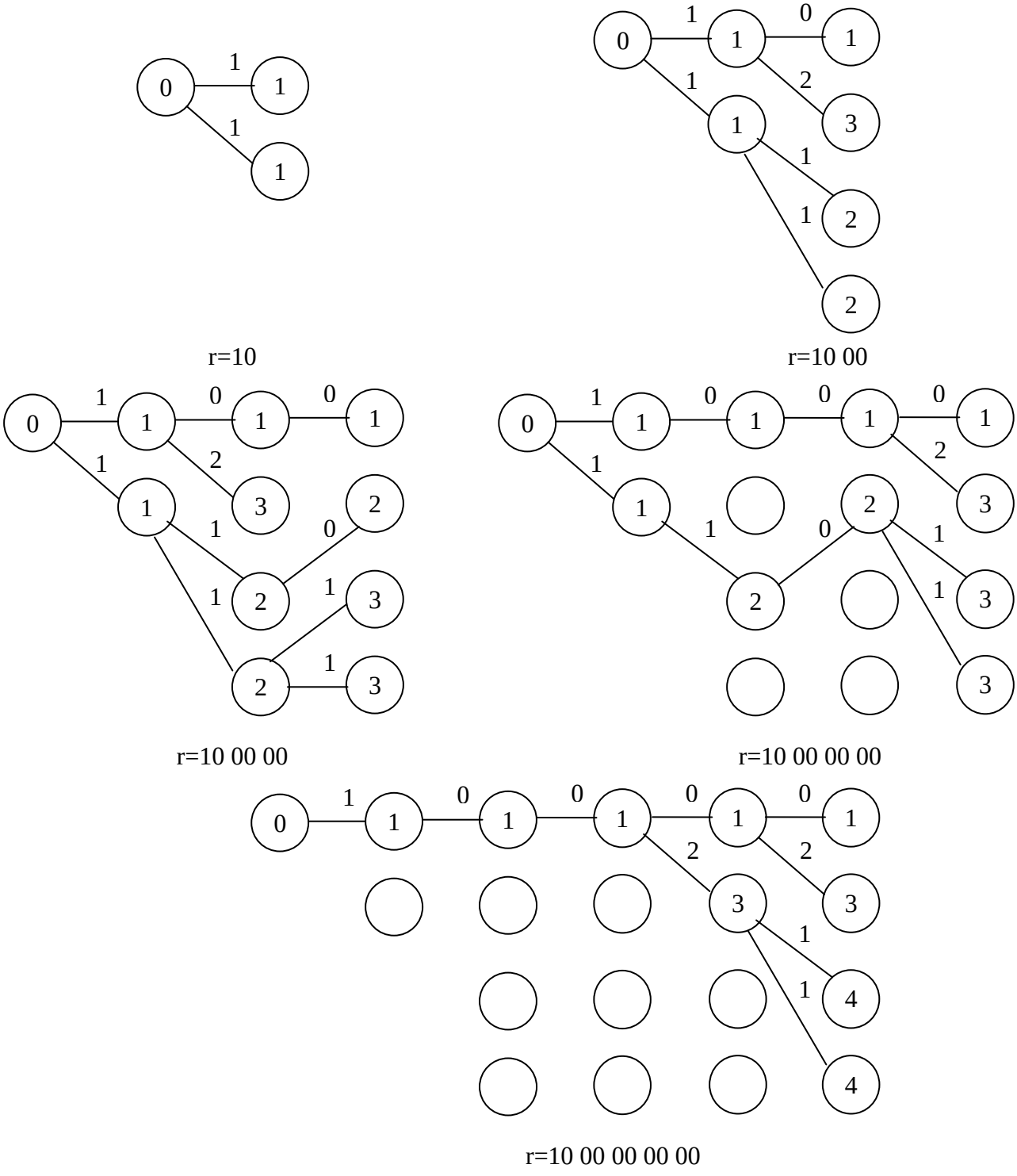


Рисунок 3.10 – Декодування методом Вітербі прийнятої послідовності

Як видно з діаграми, шляхи, що вижили, достатньо довго відрізняються один від одного. Однак на п'ятому кроці декодування перші 3 ребра всіх шляхів, що вижили, збігаються. В цей момент і приймається рішення, що перші три передані символи - 000.

Хоча уже на кроці 4 (r=10 00 00 00) відмінності метрик правильного і неправильного шляхів достатньо великі: $МШ_{пр}$=1, а $МШ_{пом}$=3-4. Тобто, можна було б обмежити глибину декодування.

Контрольні питання і вправи

1. Основні види та причини завад в комп'ютерних мережах.
2. Які способи боротьби з випадковими завадами?
3. Сформулюйте та поясніть теорему Шеннона про кодування каналу із завадами.
4. Наведіть основні принципи завадостійкого кодування.
5. Наведіть класифікацію завадостійких кодів.
6. Наведіть граничні співвідношення між параметрами завадостійких кодів.
7. Охарактеризуйте блокові корегуючі коди.
8. Які способи подання лінійних блокових кодів?
9. Що таке породжувальна і перевірна матриці?
10. Поясніть такі поняття як породжувальний і перевірний поліноми.
11. Дайте характеристику деревовидних кодів.
12. Які Вам відомі алгоритми декодування згортних кодів?
13. Охарактеризуйте алгоритм декодування Вітербі.

Вправи

1. Визначити кодову відстань d_{min} для такого коду: A_1=000000, A_2=000111, A_3=111000, A_4=111111. Відповідь: d_{min}=3.
2. Скільки помилок може виявляти і виправляти такий код: A_1=000000, A_2=000111, A_3=111000, A_4=111111. Відповідь: виявляє 2 помилки, виправляє 1 помилку.
3. Нехай передається повідомлення, що складається з восьми символів A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, які подані такими кодовими комбінаціями A_1 = 000, A_2 = 001, A_3 = 010, A_4 = 011, A_5 = 100, A_6 = =101, A_7 = 110, A_8 = 111. Побудувати код з перевіркою на парність.

Відповідь: 0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111.

4. Побудувати код Хеммінга з d_{min}=3 для наступної послідовності: 1101. Відповідь: 1010101.
5. Побудувати код Хеммінга з d_{min}=4 для наступної послідовності: 1101. Відповідь: 10101010.
6. Прийнята послідовність 1110001101 подає код Хеммінга з d_{min}=3. Визначити інформаційні символи. Відповідь: 110101.
7. Визначити кількість помилок, що виправляє код з повторенням типу (3,1). Відповідь: одну помилку.
8. Використовуючи табл. 3.2 знайти правила побудови коду (10,3), який виправляє всі одиничні та подвійні помилки (див. прикл. 3.3).
9. Використовуючи табл. 3.2 побудувати код (10,3) для наступної послідовності: 101. Відповідь: 0101110011.
10. З використанням циклічного несистематичного коду, що задається породжувальним поліномом $g(x)=1+x+x^3$, закодувати послідовність m = (0111). Відповідь: 0100011.
11. Визначити синдроми для всіх векторів помилок циклічного несистематичного коду з вправи 10. Відповідь: для 1-7 розрядів S=4, 2 1, 6, 3, 7, 5.
12. З використанням циклічного систематичного коду (див. прикл. 3.4), що задається породжувальним поліномом $g(x)=1+x+x^3$, закодувати послідовність m=(1101). Відповідь: 0001101.
13. Закодувати інформаційну послідовність k=(1001100…) з використанням згорточного кодера з імпульсної характеристики g=(11 10 11 00 00 …). Відповідь: 1110111101011100...
14. Закодувати інформаційну послідовність k=(1001100…) з використанням згортного кодера G=(7,5). Відповідь: 1110111101011100...
15. З використанням гратчастої діаграми на рис. 3.9 декодувати методом Вітербі такий згортний код: 1010111101011100... Відповідь: 1001100…
16. Розробити програму завадостійкого кодування-декодування файлів з використанням коду Хеммінга (7,4). Мова програмування С++ .
17. Розробити програму завадостійкого кодування-декодування файлів з використанням згортного коду G=(7,5). Мова програмування С++ .

4 ОСНОВИ КРИПТОЛОГІЇ

4.1 Основні означення і терміни

Проблема захисту інформації шляхом її перетворення, що виключає її прочитування сторонньою особою, хвилювала людський розум з давніх часів [28-29]. Історія криптографії - ровесниця історії людської мови. Більш того, спочатку писемність сама по собі була криптографічною системою, оскільки в стародавніх суспільствах нею володіли тільки вибрані. Священні книги Давнього Єгипту, Стародавньої Індії тому приклади.

З широким розповсюдженням писемності криптографія стала формуватися як самостійна наука. Перші криптосистеми зустрічаються вже на початку нашої ери. Так, Цезар в своєму листуванні використовував вже більш менш систематичний шифр, що отримав його ім'я.

Бурхливий розвиток криптографічні системи отримали в роки першої і другої світових воєн. Починаючи з післявоєнного часу і по нинішній день поява обчислювальних засобів прискорила розробку і вдосконалення криптографічних методів.

Чому проблема використання криптографічних методів в інформаційних системах (ІС) стала зараз особливо актуальна?

З одного боку, розширилося використання комп'ютерних мереж, зокрема глобальної мережі Інтернет, по яких передаються великі об'єми інформації державного, військового, комерційного і приватного характеру, що не допускає можливості доступу до неї сторонніх осіб.

З іншого боку, поява нових потужних комп'ютерів, технологій мережевих і нейронних обчислень зробило можливою дискредитацію криптографічних систем, що ще недавно вважалися практично не розкривними.

Проблемою захисту інформації шляхом її перетворення займається криптологія (kryptos - таємний, logos - слово). Криптологія розділяється на два напрями - криптографію і криптоаналіз. Цілі цих напрямів прямо протилежні.

Криптографія - наука про способи двунаправленого перетворення інформації з метою конфіденційної передачі її по незахищеному каналу між двома станціями, розділеними в просторі або часі. Криптографія,

використовуючи досягнення в першу чергу математики, дозволяє модифікувати дані таким чином, що ніякі найсучасніші ЕОМ за розумний період часу не можуть відновити вихідний текст, відомий тільки відправникові й одержувачеві.

Сфера інтересів криптоаналізу - дослідження можливості розшифрування інформації без знання ключів.

Наукова криптологія бере початок з роботи К. Шеннона "Теорія зв'язку в секретних системах" (1949 р.), в якій було показано, що для деякого випадкового шифру кількість знаків шифротексту, отримавши який криптоаналітик при необхідних обчислювальних ресурсах зможе відновити ключ (тобто розкрити шифр), становить:

$$n = \frac{H(Z)}{r \log N}, \qquad (4.1)$$

де H(Z) – ентропія ключа;

r – надмірність відкритого тексту;

N - обсяг алфавіту.

З виразу (4.1) видно, що зниження надмірності (стиснення даних) може значно збільшити криптостійкість навіть для коротких ключів [1].

В подальшому основна увага буде приділена криптографічним методам.

Залежно від наявності або відсутності ключа алгоритми, що шифрують, діляться на тайнопис (стеганографія – steganos – таємниця; graphy - запис) [30] і криптографію [12, 31-35]. Залежно від відповідності ключів шифрування й дешифрування - на симетричні й асиметричні [32]. Залежно від типу перетворень, що використовуються - на підставкові й переставні. Залежно від розміру блока, що шифрується, - на потокові й блокові шифри [32].

Тайнопис (стеганографія) відрізняється тим, що відправник і одержувач виконують над повідомленням перетворення, відомі тільки їм двом. Стороннім особам невідомий сам алгоритм шифрування. Деякі фахівці вважають, що тайнопис не є криптографією взагалі.

Сучасна криптографія включає чотири великі розділи:

- симетричні криптосистеми;
- криптосистеми з відкритим ключем;
- системи електронного підпису;
- управління ключами.

Основні напрями використання криптографічних методів - передача конфіденційної інформації по каналах зв'язку (наприклад, електронна пошта), встановлення достовірності передаваних повідомлень, зберігання інформації (документів, баз даних) на носіях в зашифрованому вигляді.

Отже, криптографія дає можливість перетворити інформацію таким чином, що її прочитання (відновлення) можливе тільки при знанні ключа.

Як інформація, що підлягає шифруванню і дешифруванню, розглядатимуться тексти, побудовані на деякому *алфавіті.* Під цими термінами розуміється наступне.

Алфавіт - скінченна множина знаків, що використовується для кодування інформації.

Текст - впорядкований набір з елементів алфавіту.

Як приклади алфавітів, використовуваних в сучасних ІС, можна привести такі:

- алфавіт Z33 - 32 букви російського алфавіту і пропуск;
- алфавіт Z256 - символи, що входять в стандартні коди ASCII і КОИ-8;
- бінарний алфавіт - Z2 = {0,1};
- восьмеричний алфавіт або шістнадцятковий алфавіт.

Шифрування *(encryption)* – процес перетворення: початковий текст, який носить також назву відкритого тексту, замінюється шифрованим текстом (рис. 4.1).

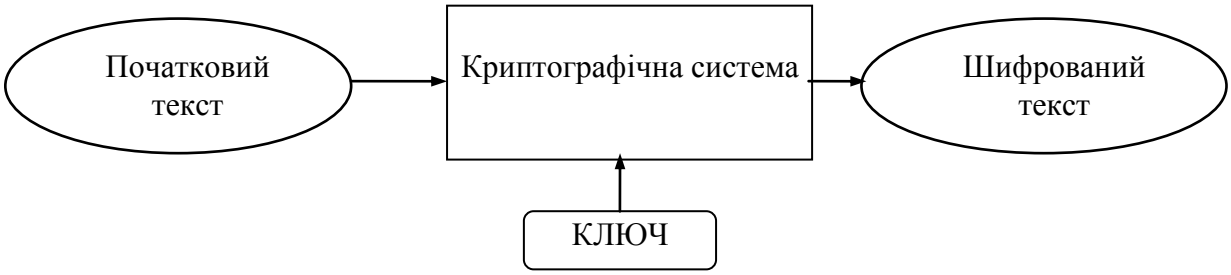


Рисунок 4.1 – Процес шифрування

Дешифрування *(decryption)*- обернений шифруванню процес. На основі ключа шифрований текст перетворюється в початковий (рис. 4.2).

Ключ *(key)* - інформація, необхідна для безперешкодного шифрування і дешифрування текстів.

Рисунок 4.2 – Процес дешифрування

Криптографічна система є сімейством **T** перетворень відкритого тексту. Члени цього сімейства індексуються, або позначаються символом *k;* параметр *k* є ключем. Простір ключів *K* - це набір можливих значень ключа. Зазвичай ключ є послідовним рядом букв алфавіту.

Криптосистеми розділяються на симетричні і з відкритим ключем.

У симетричних криптосистемах і для шифрування, і для дешифрування використовується один і той же ключ *(private key).*

У системах з відкритим ключем *(public key)* використовуються два ключі - відкритий і закритий, які математично пов'язані один з одним. Інформація шифрується за допомогою відкритого ключа, який доступний всім, а розшифровується за допомогою закритого ключа, відомого тільки одержувачеві повідомлення.

Терміни «розподіл ключів» і «управління ключами» відносяться до процесів системи обробки інформації, змістом яких є складання і розподіл ключів між користувачами.

Електронним (цифровим) підписом називається приєднуване до тексту його криптографічне перетворення, яке дозволяє при отриманні тексту іншим користувачем перевірити авторство і достовірність повідомлення.

Криптостійкістю називається характеристика шифру, яка визначає його стійкість до дешифрування без знання ключа (тобто до криптоаналізу). Є декілька показників криптостійкості, серед яких:

- кількість всіх можливих ключів;

- середній час, необхідний для криптоаналізу.

Перетворення T_k визначається відповідним алгоритмом і значенням параметра *k.* Ефективність шифрування з метою захисту інформації залежить від збереження таємниці ключа і криптостійкості шифру.

4.2 Класифікація алгоритмів шифрування

Розрізняють шифрування двох типів:

- симетричне (із секретним ключем)
- несиметричне (із відкритим ключем).

При симетричному шифруванні для шифрування і дешифрування використовується один і той же ключ (рис. 4.3). Цей ключ повинен зберігатись в тайні і передаватись способом, що виключає перехоплення. Симетричне шифрування характеризується високою швидкістю шифрування, однак ключ шифрування може бути перехоплений.

Рисунок 4.3 - Симетричне шифрування

Несиметричне шифрування складніше, але надійніше (рис. 4.4). В асиметричних алгоритмах для шифрування використовується один ключ (відкритий), а для дешифрування інший (секретний).

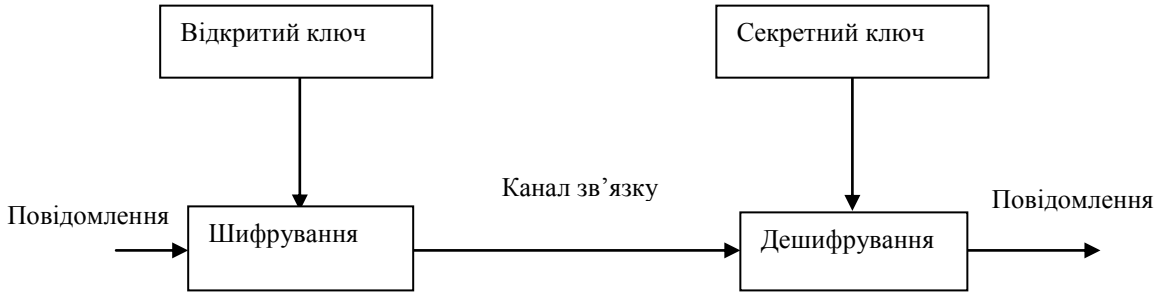


Рисунок 4. 4 - Несиметричне шифрування

Секретний ключ відомий тільки отримувачу повідомлення. Для реалізації алгоритмів з відкритими ключами, необхідно виконання

складних обчислень, тому швидкодія цих алгоритмів нижча в порівнянні з симетричними.

У випадку, якщо крипостійкість забезпечується за рахунок збереження секретності самого алгоритму шифрування, то такі алгоритми називають обмеженими. В сучасних умовах непридатні в першу чергу через велику вартість.

Симетричні алгоритми шифрування в свою чергу можна розбити на дві групи:

- потокові алгоритми шифрування
- блокові алгоритми шифрування.

В потокових шифрах кожний біт початкової інформації шифрується незалежно від інших за допомогою гамування.

При блоковому шифруванні інформація розбивається на блоки фіксованої довжини і шифрується поблоково. Довжина блока звичайно складає 64 біти. Розрізняють такі блокові шифри:

- шифри перестановки
- шифри заміни (підстановки).

Шифри перестановки переставляють елементи відкритих даних в деякому новому порядку. Розрізняють шифри вертикальної, горизонтальної та інших перестановок.

Шифри заміни замінюють елементи відкритих даних на інші елементи за визначеним правилом. Шифри заміни поділяються на дві групи:

- моноалфавітні (код Цезаря)
- поліалфавітні (шифр Відженера та ін.)

В сучасних криптографічних системах використовують як правило обидва способи шифрування (підстановки і перестановки). Такі алгоритми називаються складеними. Вони більш стійкі в порівнянні з шифрами, що використовують один якийсь спосіб. Найбільш часто використовуються такі алгоритми шифрування:

- симетричні – DES (Data Encryption Standard), IDEA (International Decryption-Encryption Algorithm), ГОСТ 28147-89 та ін.;

- несиметричні – RSA (Rivest, Shamir, Alderman), PGP (Pretty Good Privacy) [35].

4.3 Симетричні системи шифрування

Шифри Віжінера, Цезаря, Вернама. Це один з найбільш поширених шифрів. Він реалізовує багатоалфавітну підстановку. Суть цього методу полягає в тому, що кожна буква алфавіту нумерується. Наприклад, буквам англійського алфавіту ставлять у відповідність цифри від 0 (A=0) до 25 (Z=25). Ключ являє собою слово або послідовність букв, які підписують з повтореннями під повідомленням.

Цифровий еквівалент кожної букви криптограми визначається в результаті додавання з приведенням за модулем розміру алфавіту (26 в нашому прикладі) цифрових еквівалентів букви повідомлення і букви ключа, що лежить під нею (приклад 4.1).

Приклад 4.1. Зашифрувати слово CAREFULLY кодом Віжінера з ключем PIES.

Розв'язування. Запишемо букви повідомлення, розташувавши під ними їх цифрові еквіваленти.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Аналогічно внизу запишемо ключ.

C	A	R	E	F	U	L	L	Y
2	0	17	4	5	20	11	11	24
P	I	E	S	P	I	E	S	P
15	8	4	18	15	8	4	18	15

Додаючи верхні і нижні цифрові еквіваленти з приведенням за модулем 26, одержимо:

17	8	21	22	20	2	15	3	13.

Що відповідає криптограмі RIVWUCPDN.

Шифр Віжінера з ключем, що складається з однієї букви, відомий як код Цезаря, а з необмеженим ключем, що не повторюється, як код Вернама [2].

Гаммування. Гаммування є також широко вживаним криптографічним перетворенням. Насправді межа між гаммуванням і використанням нескінченних ключів і шифрів Віжінера, про які йшлося вище, вельми умовна.

Принцип *шифрування* гаммуванням полягає в генерації гамми шифру за допомогою датчика псевдовипадкових чисел і накладенні отриманої гамми на відкриті дані так, щоб можна було відновити початкові дані (наприклад, використовуючи додавання за модулем 2).

Процес *дешифрування* даних зводиться до повторної генерації гамми шифру при відомому ключі і накладенні цієї гамми на зашифровані дані.

Отриманий зашифрований текст є достатньо важким для розкриття в тому випадку, якщо гамма шифру не містить бітових послідовностей, що повторюються. По суті справи гамма шифру повинна змінюватися випадковим чином для кожного шифрованого слова. Фактично ж, якщо період гамми перевищує довжину всього зашифрованого тексту і невідома ніяка частина початкового тексту, то шифр можна розкрити тільки прямим перебором (пробій на ключ). Криптостійкість в цьому випадку визначається розміром ключа.

Метод гаммування стає безсилим, якщо зловмисникові стає відомий фрагмент початкового тексту і відповідна йому шифрограма. Простим відніманням за модулем визначається відрізок псевдовипадкової послідовності і за ним відновлюється вся послідовність. Зловмисник може зробити це на основі припущень про зміст початкового тексту. Так, якщо більшість повідомлень починаються із слів “ТАЄМНО”, то криптоаналіз всього тексту значно полегшується. Це слід враховувати при створенні реальних систем інформаційної безпеки.

Найбільш широке застосування знаходять лінійні конгруентні генератори ПВЧ. Цей генератор формує послідовність псевдовипадкових чисел $T_1, T_2, \ldots, T_m$, використовуючи співвідношення:

$$T_{i+1} = (aT_i + c) \bmod m, \qquad (4.2)$$

де a і c - константи;

T_0 – початкова величина, вибрана як породжувальне число.

Значення *m* зазвичай встановлюється рівним 2^n, *n* - довжина машинного слова в бітах.

Період повторення псевдовипадкових чисел залежить від вибраних значень a і c. Лінійний конгруентний генератор має максимальну довжину m тільки тоді, коли c - непарне і $a \bmod 4 = 1$ [12].

Приклад 4.2. Зашифрувати з використанням конгруентного генератора ПВЧ таку послідовність байтів: 65, 66, 67.

Розв'язування. Виберемо параметри генератора ПВЧ: a=21, c=13, m=256. При таких параметрах генератор буде мати максимальну довжину 256, що забезпечує зміну всіх розрядів байт повідомлення. Як T_0 введемо ключ шифру, нехай T_0 = 10. Тоді порядок шифрування такий.

1. Обчислюємо T_1 згідно з виразом (4.2):

$$T_1 = (aT_0 + c) \bmod m = (21*10 + 13) \bmod 256 = 223.$$

Зашифруємо перший байт повідомлення, взявши суму за модулем 2 чисел 65 і 223, поданих у двійковій формі:

$$\begin{array}{r} 11011111 \\ \underline{01000001} \\ 10011110 \end{array}.$$

В десятковій формі: 158.

2. Обчислюємо T_2 згідно з виразом (4.2):

$$T_2 = (aT_1 + c) \bmod m = (21*223 + 13) \bmod 256 = 88.$$

Зашифруємо другий байт повідомлення, взявши суму за модулем 2 чисел 66 і 88, поданих у двійковій формі:

$$\begin{array}{r} 01011000 \\ \underline{01000010} \\ 00011010 \end{array}.$$

В десятковій формі: 26.

3. Обчислюємо T_3 згідно з виразом (4.2):

$$T_3 = (aT_2 + c) \bmod m = (21*88 + 13) \bmod 256 = 69.$$

Зашифруємо третій байт повідомлення, взявши аналогічно суму за модулем 2 чисел 67 і 69. Отримаємо 6. Таким чином зашифроване повідомлення таке: 158, 26, 6.

Стандарт шифрування DES. Це типовий представник алгоритмів, що використовують симетричне шифрування. Отримав офіційний статус стандарту США в 1977 році. На основі **DES** розроблено міжнародний стандарт ISO 8372-87[31-32, 34-35]. Загальна схема процесу шифрування наведена на рис. 4.5. Тобто, використовується мережа Фейстеля, яка забезпечує багатократне використання ключа. Алгоритм застосовується для шифрування і дешифрування блоків даних довжиною 64 біти під управлінням 64-бітового ключа. До блока, який підлягає шифруванню,

застосовується початкова перестановка (IP), потім складна обчислювальна процедура в залежності від ключа, а в кінці обернена перестановка (IP^{-1}).

Початкова перестановка IP визначається такою таблицею:

58 50 42 34 26 18 10 02

60 52 44 36 28 20 12 04

62 54 46 38 30 22 14 06

64 56 48 40 32 24 16 08

57 49 41 33 25 17 09 01

59 51 43 35 27 19 11 03

61 53 45 37 29 21 13 05

63 55 47 39 31 23 15 07

Де елемент a(i,j) - номер біта вхідної 64-бітової послідовності.

Обернена перестановка IP^{-1} має такий вигляд:

40 08 48 16 56 24 64 32

39 07 47 15 55 23 63 31

38 06 46 14 54 22 62 30

37 05 45 13 53 21 61 29

36 04 44 12 52 20 60 28

35 03 43 11 51 19 59 27

34 02 42 10 50 18 58 26

33 01 41 09 49 17 57 25

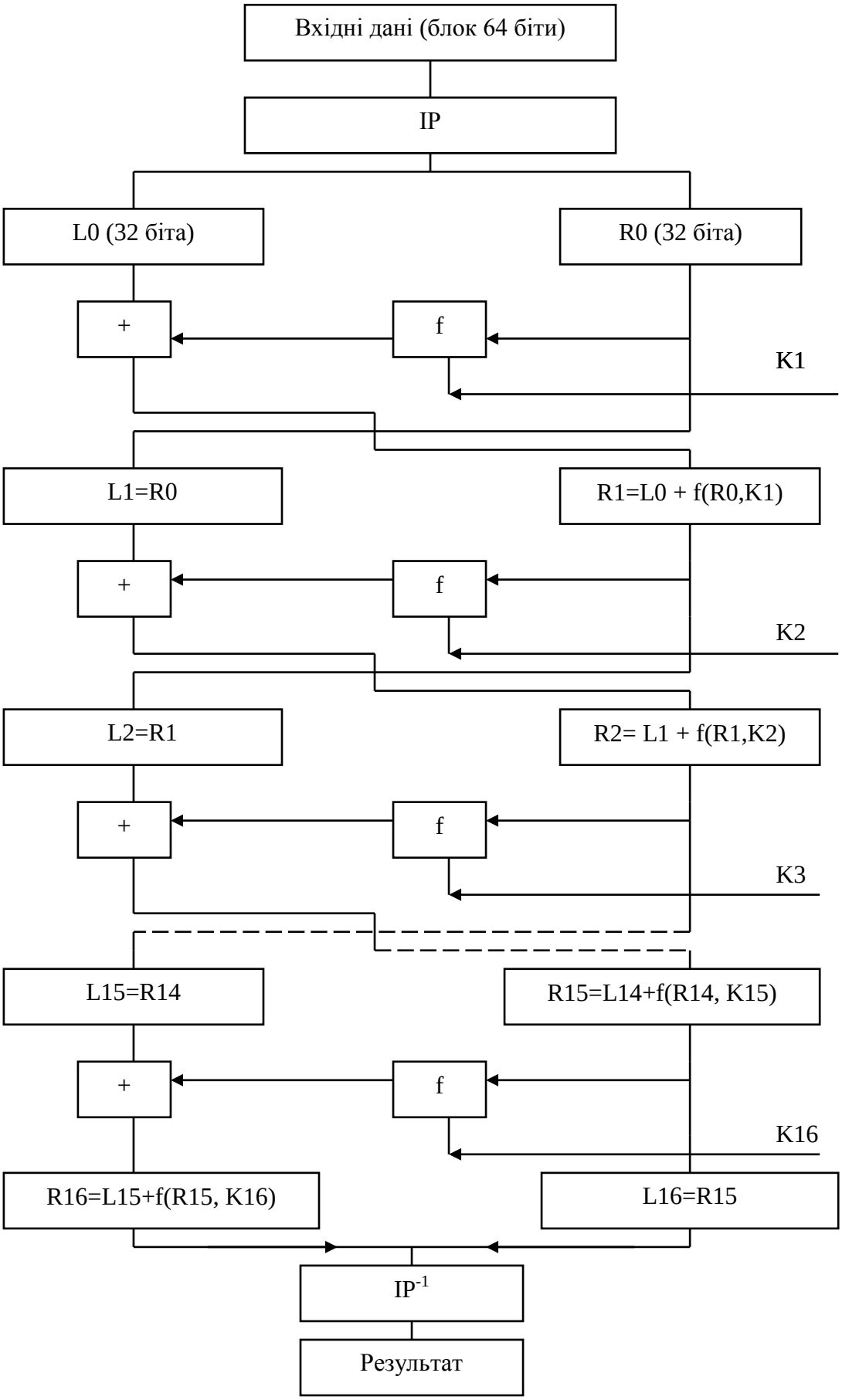


Рисунок 4.5 - Шифрування інформації згідно зі стандартом DES

L,R - 32-бітові послідовності;

K1...K16 - 48-бітові послідовності, які вибираються з 64-бітового ключа;

"+" - додавання за модулем 2.

Таким чином, процес шифрування описується такими виразами:

$$\left.\begin{aligned} L(N) &= R(N-1) \\ R(N) &= L(N-1) + f(R(N-1), K(N)) \\ K(N) &= KS(N, KEY) \end{aligned}\right\}, \qquad (4.3)$$

де KS - функція вибору ключів;

f(R,K) - функція шифрування.

Оскільки застосовується додавання за модулем 2, то з (4.3) випливає:

$$\left.\begin{aligned} R(N-1) &= L(N) \\ L(N-1) &= R(N) + f(R(N-1), K(N)) \end{aligned}\right\}. \qquad (4.4)$$

Тобто, для дешифрування необхідно застосувати той же алгоритм, що й для шифрування, тільки ключі вибираються в зворотному порядку.

Загальна схема обчислення f(R,K) наведена на рис. 4.6.

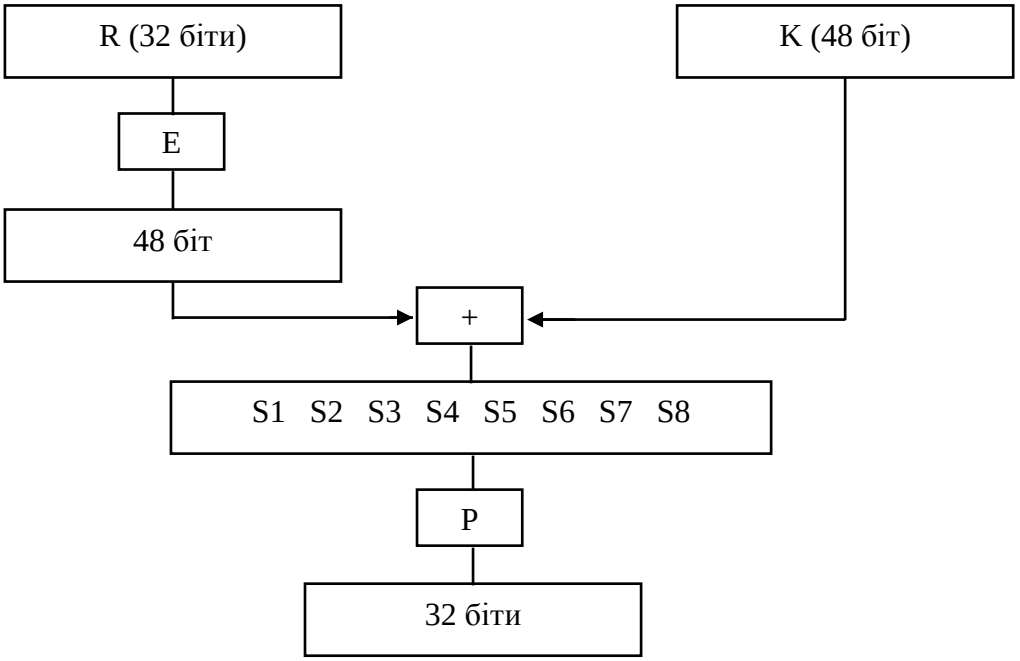


Рисунок.4.6 - Обчислення функції шифрування f(R,K)

Тут "E" позначає функцію, для якої вхідним є блок із 32 біт, а вихідним - блок із 48 біт. Вихідні 48 біт, записані у вигляді восьми

блоків по 6 біт, отримують шляхом вибору бітів із вхідних даних відповідно до такої таблиці:

Таблиця вибору бітів для "E"

32 01 02 03 04 05
04 05 06 07 08 09
08 09 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 01

Для функцій S1...S8 - вхідні дані блоки по 6 біт. Перший і останній біти цієї послідовності задають номер рядка, а 4 середніх біти номер стовпця. Наприклад, для S1:

011011	- вхідний блок;
01(1)	- номер рядка;
1101(13)	- номер стовпця;
0101(5)	- вихідні дані.

Нумерація стовпців і рядків починається з "0". Примітивні функції S1, ... , S8 такі:

S1:
14 04 13 01 02 15 11 08 03 10 06 12 05 09 00 07
00 15 07 04 14 02 13 01 10 06 12 11 09 05 03 08
04 01 14 08 13 06 02 11 15 12 09 07 03 10 05 00
15 12 08 02 04 09 01 07 05 11 03 14 10 00 06 13

S2:
15 01 08 14 06 11 03 04 09 07 02 13 12 00 05 10
03 13 04 07 15 02 08 14 12 00 01 10 06 09 11 05
00 14 07 11 10 04 13 01 05 08 12 06 09 03 02 15
13 08 10 01 03 15 04 02 11 06 07 12 00 05 14 09

S3:
10 00 09 14 06 03 15 05 01 13 12 07 11 04 02 08
13 07 00 09 03 04 06 10 02 08 05 14 12 11 15 01
13 06 04 09 08 15 03 00 11 01 02 12 05 10 14 07
01 10 13 00 06 09 08 07 04 15 14 03 11 05 02 12

S4:
07 13 14 03 00 06 09 10 01 02 08 05 11 12 04 15
13 08 11 05 06 15 00 03 04 07 02 12 01 10 14 09
10 06 09 00 12 11 07 13 15 01 03 14 05 02 08 04
03 15 00 06 10 01 13 08 09 04 05 11 12 07 02 14

S5:
02 12 04 01 07 10 11 06 08 05 03 15 13 00 14 09
14 11 02 12 04 07 13 01 05 00 15 10 03 09 08 06
04 02 01 11 10 13 07 08 15 09 12 05 06 03 00 14
11 08 12 07 01 14 02 13 06 15 00 09 10 04 05 03

S6:
12 01 10 15 09 02 06 08 00 13 03 04 14 07 05 11
10 15 04 02 07 12 09 05 06 01 13 14 00 11 03 08
09 14 15 05 02 08 12 03 07 00 04 10 01 13 11 06
04 03 02 12 09 05 15 10 11 14 01 07 06 00 08 13
S7:
04 11 02 14 15 00 08 13 03 12 09 07 05 10 06 01
13 00 11 07 04 09 01 10 14 03 05 12 02 15 08 06
01 04 11 13 12 03 07 14 10 15 06 08 00 05 09 02
06 11 13 08 01 04 10 07 09 05 00 15 14 02 03 12

S8:
13 02 08 04 06 15 11 01 10 09 03 14 05 00 12 07
01 15 13 08 10 03 07 04 12 05 06 11 00 14 09 02
07 11 04 01 09 12 14 02 00 06 10 13 15 03 05 08
02 01 14 07 04 10 08 13 15 12 09 00 03 05 06 11

Переставна функція "P" породжує 32-бітовий результат із 32-бітових вхідних послідовностей. Вона визначається такою таблицею:

Перестановка "P"

16 07 20 21
29 12 28 17
01 15 23 26
05 18 31 10
02 08 24 14
32 27 03 09
19 13 30 06
22 11 04 25

Функція вибору ключів(KS). Схема вибору 48-бітових блоків K1, ..., K16 з 64-бітового ключа наведена на рис. 4.7. А способи отримання блоків C0, D0, блоків C(n), D(n) із блоків C(n-1), D(n-1), а також ключів K1, ... , K16 визначаються таблицями, наведеними на рис. 4.8.

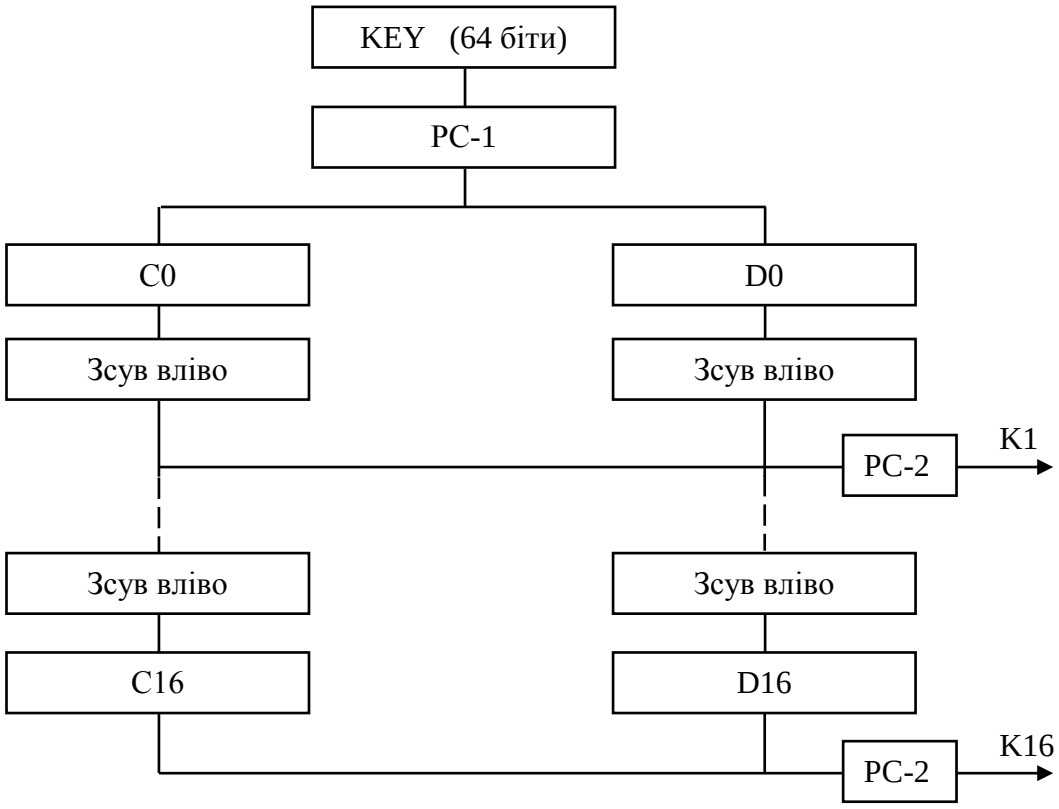


Рисунок 4.7 - Вибір ключів

Алгоритм шифрування DES давно критикують за малу довжину ключа. В січні 1999 року повідомлення зашифроване DES зламано через Інтернет за допомогою об'єднаних в єдину мережу 100 тис. комп'ютерів за 24 години. Тому виникла необхідність доробки алгоритму DES з урахуванням нових технічних можливостей. Новим стандартом став стандарт AES (Advanced Encryption Standard) з розміром ключа до 256 розрядів.

Номер ітерації	Число зсувів вліво
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Перестановка PC-1.

C0:
57 49 41 33 25 17 09
01 58 50 42 34 26 18
10 02 59 51 43 35 27
19 11 03 60 52 44 36

D0:

63 55 47 39 31 23 15
07 62 54 46 38 30 22
14 06 61 53 45 37 29
21 13 05 28 20 12 04

Примітка. Біти 8,16, ... , 64 використовуються для контролю за непарністю в кожному байті.Зсув вліво циклічний.

Перестановка PC-2

14 17 11 24 01 05
03 28 15 06 21 10
23 19 12 04 26 08
16 07 27 20 13 02
41 52 31 37 47 55
30 40 51 45 33 48
44 49 39 56 34 53
46 42 50 36 29 32

Рисунок 4.8 – Способи отримання блоків C і D та ключів

4.4 Системи з відкритим ключем

Слабким місцем симетричної криптографічної системи при практичній реалізації є проблема розподілу ключів [31-32]. Для того, щоб був можливий обмін конфіденційною інформацією між двома суб'єктами інформаційної системи (ІС), ключ повинен згенерувати один з них, а потім якимсь чином знову ж таки в конфіденційному порядку передати іншому. Тобто, в загальному випадку для передачі ключа знову ж таки потрібне використання якоїсь криптосистеми.

Для вирішення цієї проблеми на основі результатів, отриманих класичною і сучасною алгеброю, були запропоновані системи з відкритим ключом.

Суть їх полягає в тому, що кожним адресатом ІС генеруються два ключі, пов'язані між собою за певним правилом. Один ключ оголошується відкритим, а інший закритим. Відкритий ключ публікується і доступний будь-кому, хто бажає послати повідомлення адресату. Секретний ключ зберігається в таємниці.

Початковий текст шифрується відкритим ключем адресата і передається йому. Зашифрований текст в принципі не може бути розшифрований тим же відкритим ключем. Дешифрування повідомлення можливе тільки з використанням закритого ключа, який відомий тільки самому адресатові.

Криптографічні системи з відкритим ключем використовують так звані необоротні або односторонні функції, які мають таку властивість: при заданому значенні x відносно просто обчислити значення f(x), проте якщо відомо y=f(x), то немає простого шляху для обчислення значення x.

Безліч класів необоротних функцій породжує різноманітність систем з відкритим ключем. Проте не всяка необоротна функція годиться для використання в реальних ІС.

У самому означенні необоротності присутня невизначеність. Під необоротністю розуміється не теоретична необоротність, а практична неможливість обчислити зворотне значення, використовуючи сучасні обчислювальні засоби за досяжний інтервал часу.

Тому щоб гарантувати надійний захист інформації, до систем з відкритим ключем (СВК) висуваються дві важливих і очевидних вимоги.

1. Перетворення початкового тексту повинно бути необоротним і виключати його відновлення на основі відкритого ключа.

2. Визначення закритого ключа на основі відкритого також повинно бути неможливим на сучасному технологічному рівні. При цьому бажана точна нижня оцінка складності (кількості операцій) розкриття шифру.

Алгоритми шифрування з відкритим ключем набули широкого поширення в сучасних інформаційних системах. Так, алгоритм RSA став світовим стандартом де-факто для відкритих систем і рекомендований МККТТ (Міжнародний консультаційний комітет з телефонії і телеграфії).

Взагалі ж всі пропоновані сьогодні криптосистеми з відкритим ключем опираються на один з таких типів необоротних перетворень [31-32]:

- розкладання великих чисел на прості множники;
- обчислення логарифма в скінченному полі;
- обчислення коренів алгебраїчних рівнянь.

Криптосистеми з відкритим ключем (СВК) можна використовувати в трьох напрямках.

1. Як самостійні засоби захисту даних, що передаються або зберігаються.
2. Як засоби для розподілу ключів. Алгоритми СВК більш трудомісткі, ніж традиційні криптосистеми. Тому часто на практиці раціонально за допомогою СВК розподіляти ключі, об'єм яких незначний. А потім за допомогою звичайних алгоритмів здійснювати обмін великими інформаційними потоками.
3. Як засоби аутентифікації користувачів (електронний підпис).

4.5 Елементи теорії чисел

Багато криптографічних алгоритмів базуються на результатах класичної теорії чисел. Ми розглянемо необхідний мінімум із цієї теорії. Класичні теореми Ферма, Ейлера і ряд інших результатів з теорії чисел будуть наведені без доведення, які можуть бути знайдені практично в будь-якому підручнику з теорії чисел [32].

Означення 4.1. Ціле позитивне число p називається простим, якщо воно не ділиться ні на яке інше число, окрім самого себе і одиниці.

Приклад 4.3. Числа 11, 23 - прості; числа 27,33 - складені (27 ділиться на 3 і на 9, 33 ділиться на 3 і на 1).

Теорема 4.1 (основна теорема арифметики). Будь-яке ціле додатне число може бути подано у вигляді добутку простих чисел, причому єдиним чином.

Приклад 4.4. $27 = 3 \cdot 3 \cdot 3$, $33 = 3 \cdot 11$.

Означення 4.2. Два числа називаються *взаємно простими,* якщо вони не мають жодного спільного дільника окрім одиниці.

Приклад 4.5. Числа 27 і 28 взаємно прості (у них немає загальних дільників окрім одиниці), числа 27 і 33 - ні (у них є спільний дільник 3).

Означення 4.3 (функція Ейлера). Нехай дано ціле число $N > 1$. Значення функції Ейлера $\varphi(N)$ рівне кількості чисел у ряді 1,2,3..., N- 1, взаємно простих з N.

Приклад 4.6.

$\varphi(10) = ?$ 1, ~~2~~, 3, ~~4, 5, 6,~~ 7, ~~8~~, 9

$\varphi(10) = 4$

$\varphi(12) = ?$ 1, ~~2, 3, 4~~,5, ~~6,~~ 7, ~~8, 9, 10~~, 11

$\varphi(12) = 4$

(тут закреслені числа, не взаємно прості з аргументом).

Твердження 4.2. Якщо p - просте число, то $\varphi(p) = p - 1$.

Доведення. У ряду 1, 2, 3,..., p-1 всі числа взаємно прості з p, оскільки p - просте число і за означенням не ділиться ні на яке інше число.

Твердження 4.3. Нехай p і q - два різні прості числа ($p \neq q$). Тоді $\varphi(pq)=(p-1)(q-1)$.

Доведення. У ряду 1, 2..., pq-1 не взаємно прості з pq будуть числа p,2p,3p...,(q- 1)p і q, 2q, 3q...,(p-1)q.

Всього таких чисел буде (q-1) + (p-1). Отже, кількість чисел, взаємно простих з pq, буде:

$$pq - 1 - (p - 1) - (q - 1) = pq - p - q + 1 = (p-1)(q-1).$$

Теорема 4.4 (Ферма). Нехай p - просте число і $0 < a < p$. Тоді

$$a^{p-1} \bmod p = 1.$$

Приклад 4.7. $p=13$, $a = 2$

$2^{12} \bmod 13 = (2^2)^2 \cdot ((2^2)^2)^2 \bmod 13 = 3 \cdot 9 \bmod 13 = 1;$

p=11, a=10

$10^{10} \bmod 11 = 10^2 \cdot ((10^2)^2)^2 \bmod 11 = 1\text{-}1 = 1.$

Теорема 4.5 (Ейлера). Нехай a і b - взаємно прості числа. Тоді

$$a^{\varphi(b)} \bmod b = 1.$$

Теорема Ферма є окремим випадком теореми Ейлера, коли b - просте число.

Приклад 4.8. $\varphi(12) = 4$,

$5^4 \bmod 12 = (5^2)^2 \bmod 12 = (1^2)^2 \bmod 12 = 1.$

$\varphi(21) = 2 \cdot 6 = 12$,

$2^{12} \bmod 21 = 2^4 \cdot (2^4)^2 \bmod 21 = (16 \cdot 4) \bmod 21 = 1.$

Нам знадобиться ще одна теорема, близька до теореми Ейлера.

Теорема 4.6. Якщо p і q прості числа, p<>q і k - довільне ціле число, то

$$a^{k\varphi(pq)+1} \bmod (pq) = a. \qquad (4.5)$$

Приклад 4.9. Візьмемо p = 5, q = 7. Тоді pq = 35, а функція Ейлера - $\varphi(35)=(p-1)(q-1)=4 \cdot 6=24$. Розглянемо випадок k = 2, тобто підноситимемо числа до степеня $2 \cdot 24 + 1 = 49$. Отримаємо

$$9^{49} \bmod 35 = 9, \quad 23^{49} \bmod 35 = 23.$$

Оскільки кожне з чисел 9 і 23 взаємно просте з модулем 35, і за теоремою Ейлера $9^{24} \bmod 35 = 1$, $23^{24} \bmod 35 = 1$. Проте теорема 4.6 залишається правильною і для таких чисел:

$$10^{49} \bmod 35 = 10, \quad 28^{49} \bmod 35 = 28.$$

Тоді як теорему Ейлера для них не можна застосувати (кожне з чисел 10 і 28 не взаємно просте з модулем 35 і $10^{24} \bmod 35 = 15$, $28^{24} \bmod 35 = 21$).

Означення 4.4. Нехай a і b - два цілі додатні числа. Найбільший спільний дільник чисел a і b є найбільше число c, яке ділить і a, і b:

$$c = \gcd(a,b). \qquad (4.6)$$

Позначення gcd для найбільшого спільного дільника походить від англійських слів greatest common divisor і прийняте в сучасній літературі.

Приклад 4.10. gcd(10,15)= 5; gcd(8,28)=4.

Для знаходження найбільшого спільного дільника можна використовувати алгоритм, відомий як алгоритм Евкліда.

Алгоритм 4.1. Алгоритм Евкліда

ВХІД: Додатні цілі числа a,b, $a \geq b$.

ВИХІД: Найбільший спільний дільник gcd(a, b).

1. WHILE $b \neq 0$ DO
2. $r \leftarrow a \bmod b$, $a \leftarrow b$, $b \leftarrow r$.
3. RETURN a.

Приклад 4.11. Покажемо, як за допомогою алгоритму Евкліда обчислюється gcd (28,8):

a :	28	8	4
b :	8	4	0
r :	4	0	

Тут кожен стовпець є черговою ітерацією алгоритму. Процес продовжується до тих пір, поки b не стане рівним нулю. Тоді в значенні змінної a міститься відповідь (4).

Для багатьох криптографічних систем, що розглядаються в наступних розділах, актуальний так званий узагальнений алгоритм Евкліда, з яким пов'язана наступна теорема.

Теорема 4.7. Нехай a і b — два цілі додатні числа. Тоді існують цілі (не обов'язково додатні) числа x і y, такі, що

$$ax + by = \gcd(a,b). \qquad (4.7)$$

Узагальнений алгоритм Евкліда служить для знаходження gcd(a,b) і x,y, що задовольняють (4.7). Введемо три рядки $U=(u_1,u_2,u_3)$, $V=(v_1,v_2,v_3)$ і $T=(t_1,t_2,t_3)$. Тоді алгоритм записується таким чином.

Алгоритм 4.2. Узагальнений алгоритм Евкліда

ВХІД: Додатні цілі числа a, b, $a \geq b$.

ВИХІД: gcd(a,b), x, y, що задовольняють (4.7).

1. $U \leftarrow (a,1,0)$, $V \leftarrow (b,0,1)$.
2. WHILE $v_1 \neq 0$ DO
3. $q \leftarrow u_1 \operatorname{div} v_1$;
4. $T \leftarrow (u_1 \bmod v_1, u_2 - qv_2, u_3 - qv_3)$;
5. $U \leftarrow V$, $V \leftarrow T$.
6. RETURN U = (gcd(a,b),x,y).

Результат міститься в рядку U.

Операція div в алгоритмі - це цілочислове ділення

$$a \text{ div } b = [a/b].$$

Доведення коректності алгоритму 4.2 може бути знайдено в [31].

Приклад 4.12. Нехай a=28, b=19. Знайдемо числа x і y, що задовольняють (4.7).

U				28	1	0	
V	U			19	0	1	
T	V	U		9	1	-1	q=1
	T	V	U	1	-2	3	q=2
		T	V	0	19	-28	q=9

Пояснимо подану схему. Спочатку в рядок U записуються числа (28,1,0), а в рядок V - числа (19,0,1) (це перші два рядки на схемі). Обчислюється рядок T (третій рядок в схемі). Далі як рядок U береться другий рядок в схемі, а як V - третій, і знову обчислюється рядок T (четвертий рядок в схемі). Цей процес продовжується до тих пір, поки перший елемент рядка V не стане рівним нулю. Тоді передостанній рядок в схемі містить відповідь. У нашому випадку gcd(28,19)= 1, x =-2, y = 3. Виконаємо перевірку: $28 \cdot (-2) + 19 \cdot 3 = 1$.

Розглянемо одне важливе застосування узагальненого алгоритму Евкліда. У багатьох завданнях криптографії для заданих чисел c, m потрібно знаходити таке число $d < m$, що

$$c \cdot d \bmod m = 1. \qquad (4.8)$$

Відзначимо, що d існує тоді і тільки тоді, коли числа c і m взаємно прості.

Означення 4.5. Число d, що задовільняє (4.8), називається інверсією c за модулем m і часто позначається $c^{-1} \bmod m$.

Дане позначення для інверсії досить природне, оскільки ми можемо тепер переписати (4.8) у вигляді

$$cc^{-1} \bmod m = 1.$$

Множення на c^{-1} відповідає діленню на c при обчисленнях за модулем m. Аналогічно можна ввести довільні від'ємні степені при обчисленнях за модулем m:

$$c^{-e} = (c^e)^{-1} = (c^{-1})^e \pmod m.$$

Приклад 4.13. 3·4 mod 11 = 1, тому число 4 - це інверсія числа 3 за модулем 11. Можна записати $3^{-1} \bmod 11 = 4$. Число $5^{-2} \bmod 11$ може бути знайдене двома способами:

$5^{-2} \bmod 11 = (5^2 \bmod 11)^{-1} \bmod 11 = 3^{-1} \bmod 11 = 4,$

$5^{-2} \bmod 11 = (5^{-1} \bmod 11)^2 \bmod 11 = 9^2 \bmod 11 = 4.$

При обчисленнях за другим способом ми використовували рівність $5^{-1} \bmod 11 = 9$. Дійсно, $5 \cdot 9 \bmod 11 = 45 \bmod 11 = 1$.

Покажемо, як можна обчислити інверсію за допомогою узагальненого алгоритму Евкліда. Рівність (4.8) означає, що для деякого цілого k

$$cd - km = 1. \qquad (4.9)$$

Враховуючи, що c і m взаємно прості, перепишемо (4.9) у вигляді

$$m(-k) + cd = \gcd(m, c), \qquad (4.10)$$

що повністю відповідає (4.7), тут тільки по-іншому позначені змінні. Тому, щоб обчислити $c^{-1} \bmod m$, тобто знайти число d, потрібно просто використовувати узагальнений алгоритм Евкліда для розв'язання рівняння (4.10). Відмітимо, що значення k нас не цікавить, тому можна не обчислювати другі елементи рядків U, V, T. Крім того, якщо число d виходить від'ємним, то потрібно додати до нього m, оскільки за означенням число a mod m береться з множини {0,1..., m-1}.

Приклад 4.14. Обчислимо $7^{-1} \bmod 11$. Використовуємо таку ж схему запису обчислень, як в прикладі 4.12:

11	0	
7	1	
4	-1	q=1
3	2	q=1
1	-3	q=1
0	11	q=3.

Отримуємо d = -3 і d mod 11 = 11-3 = 8, тобто 7^{-1} mod 11 = 8. Перевіримо результат: 7·8 mod 11 = 56 mod 11 = 1.

Однією з найважливіших операцій в криптографії з відкритими ключами є операція піднесення до степеня за модулем. Дамо опис алгоритму виконання операції піднесення до степеня за модулем у формі, придатній для безпосередньої програмної реалізації. У назві алгоритму відображений той факт, що біти показника степеня переглядаються справа наліво, тобто від молодшого до старшого.

Алгоритм 4.3. ПІДНЕСЕННЯ ДО СТЕПЕНЯ (СПРАВА НАЛІВО)

ВХІД: Цілі числа a, $x = (x_t x_{t-1} \ldots x_0)_2$, p.

ВИХІД: Число $y = a^x \bmod p$.

1. y←1, s←a.
2. FOR i = 0,l...,t DO
3. IF $x_i = 1$ THEN y←y·s mod p;
4. s ← s·s mod p.
5. RETURN y.

Щоб показати, що за поданим алгоритмом дійсно обчислюється y, запишемо степені змінних після кожної ітерації циклу. Нехай x=100= $=(1100100)_2$, тоді:

i:	0	1	2	3	4	5	6
x_i:	0	0	1	0	0	1	1
y:	1	1	a^4	a^4	a^4	a^{36}	a^{100}
s:	a^2	a^4	a^8	a^{16}	a^{32}	a^{64}	a^{128}

У деяких ситуаціях ефективнішим виявляється алгоритм, в якому біти показника степеня переглядаються зліва направо, тобто від старшого до молодшого.

Алгоритм 4.4. ПІДНЕСЕННЯ ДО СТЕПЕНЯ (ЗЛІВА НАПРАВО)

ВХІД: Цілі числа a, $x = (x_t x_{t-1} \ldots x_0)_2$, p.

ВИХІД: Число $y = a^x \bmod p$.

1. y←1.
2. FOR i = t, t-1,...,0 DO
3. y ← y·y mod p;
4. IF $x_i = 1$ THEN y←y·a mod p;
5. RETURN y.

Щоб переконатися в тому, що алгоритм 4.4 обчислює те ж саме, що і алгоритм 4.3, запишемо степені змінної y після кожної ітерації циклу для x=100:

i:	6	5	4	3	2	1	0
x_i:	1	1	0	0	1	0	0
y:	a	a^3	a^6	a^{12}	a^{25}	a^{50}	a^{100}

Наведених в даному розділі відомостей з теорії чисел буде досить для опису основних криптографічних алгоритмів і методів.

4.6 Алгоритм шифрування RSA

Найпершою криптографічною системою з відкритим ключем, з тих що були запропоновані у відкритій літературі взагалі, є криптосистема Рівеста (Rivest R.), Шаміра (Shamir A.) і Едлемана (Adleman L.) [31]. Вона стала відома під назвою RSA або MIT криптосистема. Ця криптосистема грунтується на вірі в те, що модульне піднесення до степеня при фіксованій експоненті і модулі є однонаправленою (односторонньою) функцією з потайним ходом.

Першим етапом будь-якого асиметричного алгоритму є створення пари ключів - відкритого і закритого. Для алгоритму RSA етап створення ключів включає такі операції.

1. Вибираються два дуже великі прості числа p і q.
2. Обчислюється добуток n=p·q (модуль алгоритму).
3. Вибирається довільне ціле число $0<e<(p-1)\cdot(q-1)$, але не рівне p або q, яке є взаємно простим з $\varphi(p\cdot q)=(p-1)\cdot(q-1)$, тобто $\gcd(e,(p-1)\cdot(q-1))=1$. Як правило e беруть рівним 3, 17 або 65537 [32].
4. Визначають таке ціле число d>0, для якого істинним є таке співвідношення: $(e\cdot d) \bmod[(p-1)\cdot(q-1)]=1$ або $e\cdot d+(p-1)\cdot(q-1)\cdot y=1$. Невідомі змінні d і y знаходять з використанням узагальненого алгоритму Евкліда. Якщо в результаті обчислення згідно з алгоритмом Евкліда отримаємо d<0, то до нього необхідно додати $(p-1)\cdot(q-1)$, тобто у цьому випадку $d=d+(p-1)\cdot(q-1)$.
5. Два числа (e, n) — публікуються як відкритий ключ.
6. Число d зберігається в строгому секреті — це і є закритий ключ, який дозволить читати всі послання, зашифровані за допомогою пари чисел (e, n).

Шифрування виконується так.

1. Відправник розбиває повідомлення на блоки довжиною $k=[\log_2 n]$ біт, де квадратні дужки означають взяття цілої частини від дробового числа.
2. Подібний блок можна інтерпретувати як число m_i в діапазоні від 0 до 2^{k-1}, тобто $m_i<n$. Для кожного такого числа обчислюється вираз:

$$c_i=(m_i)^e \bmod n, \tag{4.11}$$

де c_i – зашифроване повідомлення.

Щоб розшифрувати це повідомлення необхідний секретний ключ:

$$m_i=(c_i)^d \bmod n. \tag{4.12}$$

Дійсно, оскільки m_i взаємно просте з n, то згідно з теоремою Ейлера:

$$m_i^{(p-1)(q-1)} \bmod n =1. \tag{4.13}$$

Піднесемо ліву і праву частину даної рівності до степеня y:

$$m_i^{(p-1)(q-1)y} \bmod n =1. \tag{4.14}$$

А тепер помножимо ліву і праву частини на m_i:

$$m_i^{(p-1)(q-1)y+1} \bmod n =m_i. \tag{4.15}$$

Але згідно з пунктом 4 етапу створення ключів:

$$(p-1)(q-1)y+1=ed.$$

Тобто:

$$m_i^{ed} \bmod n =m_i. \tag{4.16}$$

Замінимо у виразі (4.12) c_i його значенням і з урахуванням виразу (4.16) отримаємо:

$$m_i=(c_i)^d \bmod n=(m_i^e)^d \bmod n = m_i^{ed} \bmod n=m_i. \tag{4.17}$$

Що і потрібно було довести.

Приклад 4.15. Зашифрувати букви англійського алфавіту.

Розв'язування. Букв в англійському алфавіті 26. Згенеруємо відкритий і секретний ключі.

1. Вибираємо p=3 і q=11.
2. Обчислюємо добуток $n=p\cdot q =3\cdot 11=33$ (модуль алгоритма).
3. Вибираємо довільне число e, яке є взаємно простим з $(p-1)\cdot(q-1)=2\cdot 10=20$, тобто gcd(e, 20)=1: e=7.

4. Визначаємо таке число d, для якого істинним є співвідношення: $(e \cdot d) \bmod [(p-1) \cdot (q-1)]=1$ або $(7 \cdot d) \bmod 20 =1$. Невідомі змінні d і у знаходимо з використанням узагальненого алгоритму Евкліда: d=3.
5. Два числа (e, n)=(7,33) - відкритий ключ.
6. Числа (d, n)= (3, 33) - закритий (секретний) ключ.

Нехай цифрові еквіваленти букв такі:

A	B	C	D	…	Z
4	5	6	7		29.

Для їх подання потрібні блоки довжиною 5 біт, що відповідає вимозі на довжину блока $k=[\log_2 n]= [\log_2 33]=5$ біт.

Зашифруємо наприклад букву А:

$$c_i=(m_i)^e \bmod n=4^7 \bmod 33 = 16384 \bmod 33 = 16.$$

Для дешифрування використаємо вираз ():

$$m_i=(c_i)^d \bmod n = 16^3 \bmod 33 = 4.$$

Відновлене значення відповідає цифровому еквіваленту букви А.

Операції піднесення великих чисел до степеня достатньо трудомістка для сучасних процесорів. Тому звичайно текст повідомлення шифрується симетричним шифром з використанням ключа сеансу, а сам ключ шифрується асиметричним алгоритмом за допомогою відкритого ключа отримувача повідомлення і поміщається у файл, що передається.

Криптоалгоритм RSA – частина багатьох світових стандартів шифрування таких як ISO 9796, S/MINE та інші.

Недоліком алгоритму є те, що якщо будуть знайдені ефективні методи розкладу n на співмножники p і q, то можна буде отримати секретний ключ d за прийнятний час. Тому автори RSA рекомендують використовувати такі розміри модуля n:

- 768 біт - для приватних осіб;
- 1024 біт - для комерційної інформації;
- 2048 біт - для особливо секретної інформації [32].

4.7 Шифрування даних при ущільненні

Донедавна алгоритми ущільнення даних [12] і криптографічного захисту розвивались окремо, що призводило до значних обчислювальних витрат, оскільки при передачі і зберіганні файлів виникає необхідність в подвійному перетворенні інформації - спочатку ущільнення, а потім шифрування ущільненого файла. Тому актуальною є розробка таких

алгоритмів шифрування даних, які б за один прохід виконували шифрування інформації з її одночасним ущільненням.

Головним критерієм при виборі алгоритму ущільнення для шифрування даних є мінімум затрат на адаптацію його до розв'язування нових задач. С цієї точки зору заслуговують на увагу алгоритми ущільнення, які формують масиви символів перед виконанням ущільнення, що може бути використано при реалізації алгоритму шифрування. Іншими важливими критеріями є адаптивність алгоритму ущільнення, коефіцієнт ущільнення, простота технічної реалізації. Характеристики основних методів ущільнення за даними критеріями наведені в табл. 4.1.

Як видно з таблиці серед розглянутих алгоритмів ущільнення найбільші переваги, з точки зору застосування їх до шифрування даних, мають два алгоритми:

- ущільнення методом MTF (Move To Front);
- імовірнісний метод ущільнення.

Ці алгоритми передбачають формування таблиці символів перед виконанням ущільнення даних, яка може бути сформована за ключем шифру з використанням, наприклад, генератора псевдовипадкових чисел. До того ж ці алгоритми є адаптивними, тобто не вимагають передачі додаткової інформації, яка могла бути використана зловмисниками для злому шифру, а також характеризуються простою технічною реалізацією.

Таблиця 4.1 – Характеристики методів ущільнення

Метод ущільнення	Коефіцієнт ущільнення	Обчислювальні затрати	Адаптивний	Додаткові обчислювальні затрати для шифрування
Словниковий	Близький до оптимального для великих масивів	Середні	Так	Так
Хаффмана	Оптимальний	Середні	Ні	Так
MTF	Близький до оптимального	Середні	Так	Ні
Імовірнісний	Близький до оптимального	Малі	Так	Ні
Арифметичний	Найбільший	Великі	Так	Так

Для прикладу розглянемо більш детально шифрування і ущільнення даних методом MTF (кодування за ступенем новизни). При ущільненні інформації цим методом можливе одночасне виконання шифрування методом багатоалфавітної підстановки без додаткових затрат або із застосуванням інших методів і їх комбінацій з незначними додатковими затратами. Нехай передається повідомлення:

ω **=ABCDDAAACAB**

в алфавіті **AL={A,B,C,D}**. При появі в слові ω чергової букви "A" по лінії зв'язку передається монотонний код номера позиції, яку займає в даний момент символ "A" в списку, а символ "A" переміщується на початок списку.

Таблиця 4.2 - Кодування за ступенем новизни

Вхід	Вихід 1	Алфавіт 1	Вихід 2	Алфавіт 2
A	0_2	{ABCD}	110_2	{CBAD}
B	10_2	{ABCD}	110_2	{ACBD}
C	110_2	{BACD}	110_2	{BACD}
D	111_2	{CBAD}	111_2	{CBAD}
D	0_2	{DCBA}	0_2	{DCBA}
A	111_2	{DCBA}	111_2	{DCBA}
A	0_2	{ADCB}	0_2	{ADCB}

Таким чином символи, які часто зустрічаються, завжди будуть знаходитись на початку списку і відповідно подаватись короткими кодовими комбінаціями. Порядок кодування за ступенем новизни демонструє табл. 4.2.

Якщо список алфавіту "AL" згенерувати, наприклад, використовуючи генератор ПВЧ, то вихідна послідовність стане іншою (табл. 4.2 - вихід 2, алфавіт 2). Не знаючи початкового стану алфавіту, неможливо дешифрувати повідомлення. При цьому реалізується шифрування методом багатоалфавітної підстановки і стиснення даних одночасно. Для одночасного стиснення і шифрування даних методом багатоалфавітної підстановки може використовуватись така схема.

1. Згідно з формулою (4.2) генерується алфавіт повідомлення. Оскільки символи в комп'ютерних системах подані восьмибітовими комбінаціями, то m=256, а константи a, c і породжувальне число T_0 можна вводити як ключ шифру.

2. Виконується стиснення згідно з наведеним методом.

Додаткові затрати відсутні, оскільки генерація символів алфавіту виконується в будь-якому випадку.

При незначних додаткових затратах можливо застосувати комбінацію методів, яка включає два основні етапи.

Етап1. Кодування за ступенем новизни для стиснення і шифрування методом багатоалфавітної підстановки.

Етап2. Шифрування за допомогою додаткового генератора ПВЧ з m, рівним довжині файла, який отриманий в результаті виконання етапу 1. Цей або інший генератор ПВЧ може бути використаний також для виконання перестановок, оскільки він генерує всі числа в межах довжини початкового файла. Якщо в стисненні даних немає потреби, то ця перестановка дозволить замаскувати стиснені дані в межах файла такого ж розміру як і початковий, з попередньо записаною випадковою інформацією.

Аналіз таких відомих алгоритмів стиснення інформації як кодування Хаффмана та LZW показує також, що їх легко адаптувати до одночасного виконання стиснення і шифрування інформації. А при деяких додаткових обчисленнях згідно з формулою (4.2), ще і шифрування методом ПВЧ. Для цього достатньо лише сформувати таблицю перекодування кожного зчитаного з пам’яті символу. Ця таблиця може бути сформована, наприклад, за допомогою генератора ПВЧ з m=256. При цьому зчитаний з початкового файла символ даних задає позицію в таблиці перекодування, з якої вибирається код підстановки.

4.8 Комп’ютерна стеганографія

Стрімкий розвиток цифрових технологій та засобів телекомунікацій стимулює створення різноманітних методів захисту інформації.

Принцип стеганографії - приховування одного масиву інформації в іншому - в останні роки знайшов широке застосування. Комп'ютерні файли (зображення, звукові записи і т.п.) мають області, що не використовуються або не є важливими (наприклад, молодші розряди в цифровому поданні значення пікселя зображення або відліку звуку), й, відповідно, існує можливість заміщати їх необхідною спеціальною інформацією. Розвиток інформаційних технологій

привів до створення цілого класу методів і програмних продуктів, що дозволяють розмістити приховану інформацію у цифрові документи практично всіх типів [30].

Відомо, що для гарантованого захисту вмісту повідомлення існує два різних підходи.

Перший - це блокування несанкціонованого доступу до інформації шляхом шифрування повідомлення. Для цієї мети використовуються криптографічні методи захисту. У криптограмах, як правило, відсутні структура і закономірності, які властиві відкритим текстам. Тому, при проведенні моніторингу мереж телекомунікацій, вони легко автоматично виділяються з інформаційного потоку.

Другий підхід полягає в тому, що повідомлення, яке передається, намагаються приховати так, що б його неможливо було знайти. Для приховування факту існування інформації застосовуються стеганографічні методи захисту, які значно знижують ймовірність її виявлення. На відміну від криптографічного захисту, коли в «зловмисника» існує можливість знайти, перехопити та зробити спробу дешифрувати криптограму, стеганографічні методи дозволяють вмонтувати інформацію, що передається, в невинні на вигляд послання так, щоб не можна було навіть запідозрити існування підтексту. Шанси знайти приховане повідомлення - невеликі, але на той випадок, якщо повідомлення все-таки буде виявлено, його можна ще додатково зашифрувати. У цьому випадку стеганографія являє собою більш високий рівень захисту інформації в порівнянні з методами криптографії.

Стеганографія - це мистецтво і наука організації зв'язку, при якій приховується, власне, наявність самого зв'язку.

В найближчі роки інтерес до стеганографії буде підсилюватися. Основна передумова для цього сформована вже сьогодні. Це стрімкий розвиток комп'ютерної мережі загального користування Інтернет з такими невирішеними і суперечливими проблемами, як захист авторських прав, захист прав на особисту таємницю, організація електронної торгівлі, протиправна діяльність хакерів і терористів.

Незважаючи на те, що стеганографія як спосіб приховування секретних даних відома вже протягом тисячоріч [30], комп'ютерна стеганографія - молодий напрямок, що розвивається. Роком становлення

комп'ютерної стеганографії як науки можна вважати 1996 рік, коли була введена єдина термінологія. Нижче наведемо деякі означення.

Стеганографічна система (стегосистема) - сукупність засобів та методів, які використовуються для формування прихованого каналу передавання інформації.

Контейнер - будь-яка інформація, призначена для приховування таємних повідомлень Порожній контейнер - контейнер без вбудованого повідомлення; наповнений контейнер (стеганоконтейнер), контейнер, що містить вбудовану інформацію.

Вбудоване (приховане) повідомлення - повідомлення, що вбодовується в контейнер.

Стеганографічний канал – канал передавання прихованого повідомлення.

Стегоключ - секретний ключ, необхідний для приховування інформації. У залежності від кількості рівнів захисту (наприклад, повідомлення попередньо шифрується) у стегосистемі може бути один або кілька стегоключів.

Сьогодні стеганографічні технології активно використовуються для вирішення таких основних задач:

- захисту конфіденційної інформації від несанкціонованого доступу;
- захисту авторських прав на деякі види інтелектуальної власності;
- подолання систем моніторингу і керування мережними ресурсами;
- камуфляжу програмного забезпечення;
- створення прихованих від законного користувача каналів витоку важливої інформації.

Використання стеганографічних систем є найбільш ефективним при вирішенні проблеми захисту конфіденційної інформації. Так, наприклад, тільки одна секунда оцифрованого звуку з частотою дискретизації 44100 Гц і рівнем відліку 8 біт у стереорежимі дозволяє приховати за рахунок заміни молодших розрядів на приховуване повідомлення близько 10 Кбайт інформації. При цьому зміна значень відліків складає менше 1 %. Така зміна практично не виявляється при прослуховуванні файла більшістю людей. Приховування впроваджуваних даних, які у більшості випадків мають великий об'єм, висуває серйозні вимоги до контейнера,

розмір контейнера в декілька разів повинен перевищувати розмір даних, що вбудовуються.

Крім прихованого передавання повідомлень, стеганографія є одним із найперспективніших напрямків для автентифікації і маркування авторської продукції (цифрові водяні знаки) з метою захисту авторських прав на цифрові об'єкти від піратського копіювання. На комп'ютерні графічні зображення, аудіопродукцію, літературні твори (програми в тому числі) наноситься спеціальна мітка, що залишається невидимою для очей, але розпізнається спеціальним програмним забезпеченням. Мітка містить приховану інформацію, що підтверджує авторство. Наприклад, дані про автора, дату і місце написання твору, номери документів, що підтверджують авторство і т.п. Такі спеціальні відомості можуть розглядатися як докази при розгляді спорів про авторське право або для доказу нелегального копіювання. Основні вимоги, які висуваються до цифрових водяних знаків, є надійність і стійкість до перекручувань.

Можливість подолання систем моніторингу пов'язана в першу чергу з фактом приховування інформації в типових файлах, що не може бути автоматично виділено з інформаційного потоку.

Прикладом маскування програмного забезпечення є приховування небажаного програмного забезпечення у файлах мультимедіа.

При створенні прихованого від законного користувача каналів витоку важливої інформації використовуються особливості організації обчислювальної системи. Наприклад, у файловій системі NTFS розмір кластерів на дисках 4096 байт. Кластер - це мінімальна логічна одиниця пам'яті на дисках, якою маніпулює операційна система. Це означає, що при розмірі файла 1024 байти йому виділяється 4096 байт дискового простору. Таким чином, невикористане місце може бути відведено для приховування додаткової інформації без відображення цього факту в системних каталогах.

Більш детально питання пов'язані з комп'ютерною стеганографією розглядаються в [30].

Контрольні питання і вправи

1. Які методи захисту інформації застосовують в ОС?
2. Що таке матриця доступу?
3. Поясніть застосування списків доступу.
4. Які недоліки методів захисту інформації за допомогою паролів, матриць доступу та списків доступу?
5. Які питання розглядає криптологія?
6. Чим відрізняється криптографія від криптоаналізу?
7. Хто є основоположником наукової криптології?
8. Які основні розділи включає криптографія?
9. Наведіть класифікацію алгоритмів шифрування інформації.
10. Охарактеризуйте шифри Віжінера, Цезаря, Вернама.
11. В чому різниця між DES і RSA схемами шифрування?
12. Поясніть принципи роботи алгоритму DES.
13. Поясніть принципи роботи алгоритму RSA.
14. Поясніть порядок генерацію відкритого і секретного ключів в RSA.
15. Поясніть роботу узагальненого алгоритму Евкліда при генерації ключів в RSA.
16. Наведіть алгоритм виконання операції піднесення до степеня за модулем справа наліво.
17. Наведіть алгоритм виконання операції піднесення до степеня за модулем зліва направо.
18. Поясніть принципи шифрування інформації за допомогою генератора ПВЧ.
19. Які алгоритми ущільнення найбільш прийнятно використовувати для шифрування даних?
20. Охарактеризуйте комп'ютерну стеганографію.
21. Дайте порівняльний аналіз алгоритмів шифрування інформації.

Вправи

1. Дешифрувати шифротекст RIVWUCPDN, отриманий за допомогою шифру Віжінера з ключем PIES (див. прикл. 4.1). Відповідь: CAREFFULY.

2. Дешифрувати шифротекст 158, 26, 6, отриманий в прикладі 4.2. Відповідь: 65, 66, 67.
3. Зашифрувати текст з використанням шифру Віжінера (див. прикл. 4.1) і ключа VIGINERE : QRSTUVWXYZ. Відповідь: LZYBGZNBTG.
4. Зашифрувати з використанням конгруентного генератора ПВЧ таку послідовність байт: 170, 85, 255. Параметри генератора ПВЧ такі: a=9, c=11, m=256, T_0=10. Відповідь: 207, 205, 156.
5. За допомогою алгоритму Евкліда знайти gcd (21, 12), gcd (35, 15), gcd (45, 27), gcd (33, 16). Відповідь: 3, 5, 9, 1.
6. Визначити з використанням алгоритму Евкліда, які з пар чисел є взаємно прості: (25, 12), (25, 15), (13, 39), (40, 27). Відповідь: (25, 12), (40, 27).
7. За допомогою узагальненого алгоритму Евкліда знайти значення x і y в рівняннях:

$$21x + 12y = \gcd(21, 12);$$
$$35x + 15y = \gcd(35, 15);$$
$$45x + 27y = \gcd(45, 27);$$
$$33x + 16y = \gcd(33, 16).$$

   Відповідь: 3, -1, 2; 5, 1, -2; 9, -1, 2; 1, 1, -2.
8. Обчислити, використовуючи швидкі алгоритми піднесення до степені, такі вирази: $2^8 \bmod 10$, $3^7 \bmod 10$, $7^{19} \bmod 100$, $7^{57} \bmod 100$. Відповідь: 6, 7, 43, 7.
9. Знайти відкритий і секретний ключі в RSA, якщо p=11, q=17. Відповідь: e=13, d=37.
10. Дешифрувати методом RSA повідомлення c_i=23, якщо p=5, q=11, e=3. Відповідь: m_i=12.
11. Зашифрувати методом RSA повідомлення m_i=15, якщо p=7, q=11, e=13. Відповідь: c_i=64.
12. Розробити програму шифрування-дешифрування файлів методом Віжінера. Ключ шифру 8 байт. Мова програмування C++ .
13. Розробити програму шифрування-дешифрування файлів методом гаммування. Ключ шифру 2 байти. Мова програмування C++ .
14. Розробити програму шифрування-дешифрування файлів методом RSA. Мова програмування C++ .

5 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ І ІНФОРМАЦІЙНА БЕЗПЕКА

Захист інформації від несанкціоновного доступу включає різноманітні аспекти не пов’язані безпосередньо з кодуванням і криптографією, тобто з предметом даного курсу. Однак розробники програмного забезпечення повинні мати уявлення про загрози, які формуються ними ж при недотриманні вимог безпеки при розробці того чи іншого програмного забезпечення, наприклад, системи криптографічного захисту. Це і стало причиною включення цього розділу в даний посібник. Більш детально ці питання вивчаються на інших курсах дивись, наприклад [36-38].

5.1 Огляд сучасного програмного забезпечення

Сучасне програмне забезпечення (ПЗ), яке випускається навіть гігантами індустрії ПЗ, характеризується помилками, що з'являються і зникають від версії до версії і непередбаченими можливостями, які зловмисники використовують в своїх цілях.

Програмне забезпечення включає два основних компоненти:

- операційні системи (ОС);
- прикладне програмне забезпечення.

Операційні системи є основною частиною програмного комплексу ЕОМ і при цьому виконують величезний набір проміжних операцій між прикладними програмами і апаратним забезпеченням. Природно, що помилки в них цікавлять зловмисників понад усе. Від рівня реалізації політики безпеки в кожній конкретній ОС багато в чому залежить і загальна безпека інформаційної системи. У більшості ОС є механізми ідентифікації користувача, які забезпечують той чи інший рівень захисту інформації. Основні методи захисту інформації від несанкціонованого доступу в ОС такі:

- захист інформації за допомогою матриці управління доступом та списків управління доступом;
- захист інформації за допомогою "паролів";
- захист інформації за допомогою криптографічних методів.

Недоліки двох перших методів полягають у тому, що "ключі" доступу зберігаються в самій системі. Це може призвести до того, що

підготовлений недобросовісний користувач може їх розкрити і скористатись секретною інформацією. А при шифруванні ключ доступу не зберігається у системі. Користувач вводить його тільки тоді, коли зашифровує або розшифровує інформацію. Питання шифрування-дешифрування інформації є предметом криптології (див. розділ 4).

Розглянемо сучасні операційні системи з точки зору інформаційної безпеки. В першу чергу нас цікавитимуть безпечне розділення оперативної пам'яті і файлів між процесами і користувачами і стійкість ОС до мережевих атак.

Операційна система (ОС) MS-DOS є однозадачною ОС реального режиму мікропроцесора Intel, а тому тут не може йти мова про розділення оперативної пам'яті між процесами. Всі резидентні програми і ОС використовують загальний простір ОЗП. Захист файлів відсутній, про мережеву безпеку важко сказати що-небудь певне, оскільки на тому етапі розвитку ПЗ драйвери для мережевої взаємодії розроблялися не фірмою MicroSoft, а сторонніми розробниками.

Сімейство операційних систем Windows 95, 98, Millenium – це клони, орієнтовані на роботу в домашніх ЕОМ. Ці операційні системи використовують рівні привілеїв захищеного режиму, але не підтримують систему дескрипторів безпеки. В результаті цього будь-який додаток може отримати доступ до всього об'єму доступної оперативної пам'яті як з правами читання, так і з правами запису. Заходи мережевої безпеки присутні, проте їх реалізація не на висоті. Більш того, у версії Windows 95 була допущена грунтовна помилка, що дозволяла віддалено буквально за декілька пакетів приводити до "зависання" ЕОМ, що також значно підірвало репутацію ОС, в подальших версіях було зроблено багато кроків щодо поліпшення мережевої безпеки.

Покоління операційних систем Windows NT, 2000, XP вже значно надійніша розробка компанії MicroSoft. Дані ОС активно використовують можливості захищеного режиму процесорів Intel, і можуть надійно захистити дані і код процесу від інших програм, якщо тільки він сам не захоче надавати до них додаткового доступу ззовні процесу. За довгий час розробки було враховано безліч різних мережевих атак і помилок в системі безпеки. Виправлення до них виходили у вигляді блоків оновлень (англ. service pack). Таким чином ці операційні системи мають надійну систему мережевої безпеки.

UNIX-подібні операційні системи розроблялися як мережеві багатокористувацькі системи, а тому відразу ж містили в собі засоби інформаційної безпеки. Практично всі широкопоширені клони UNIX пройшли довгий шлях розробки і модифікацій, врахувючи всі відкриті за цей час способи атак. Достатньо добре себе зарекомендували : LINUX (S.U.S.E.), OPENBSD, FREEBSD, Sun Solaris. Основні помилки в цих системах відносяться вже не до ядра, яке працює бездоганно, а до системних і прикладних утиліт. Наявність помилок в них часто приводить до втрати всього запасу міцності системи.

Основною проблемою прикладного ПЗ є розробка його фірмами, які просто не в змозі виділити асигнування на ретельну перевірку свого продукту. Помилки в прикладному програмному забезпеченні були і залишаються основним шляхом проникнення зловмисника як на сервери, так і на робочі станції. Об'єктивна причина цього – розробка подібного ПЗ різними групами розробників, які просто не в змозі приділити належної уваги мережевій і локальній безпеці свого продукту. І якщо фірми-розробники операційних систем витрачають величезні суми на ретельні випробування поведінки їх програм в нестандартних ситуаціях, а також активно враховують багаторічний досвід своїх же помилок, то для невеликих фірм це просто не під силу, та і украй невигідно економічно.

Помилки активно шукаються групами "хакерів" практично у всьому більш менш поширеному ПЗ, проте, найбільшої популярності набувають, звичайно, дослідження програм, встановлених майже у кожного користувача. Так, наприклад, в одній з недавніх версій MicroSoft Internet Explorer була виявлена помилка, пов'язана з переповнюванням буфера, яка призводила до того, що частина URL-адреси потрапляла на "виконання" і трактувалася як послідовність команд процесора. При цьому довжини цієї ділянки вистачало, наприклад, для того, щоб завантажити на ЕОМ з мережі троянську програму і передати їй управління. У подальшій версії помилка була виправлена. Програма ICQ – найпопулярніший електронний пейджер в мережі Інтернет – в черговій своїй версії була забезпечена своїми творцями можливістю підтримувати мініатюрний WWW-сервер. Проте помилка в його реалізації дозволяла при додаванні зліва крапок в імені першого каталога діставати доступ до всіх файлів жорсткого диска - відкривався повний мережевий доступ до читання.

Багато атак використовують не тільки безпосередні помилки в реалізації ПЗ, але і непродумані розробниками аспекти використання стандартних можливостей програми. Так, мабуть, найяскравішим прикладом цього є MACRO-віруси в документах системи MicroSoft Office. Можливість виконання макросів була вбудована в цю систему з метою можливості її адаптації під користувача, але той факт, що макроси можуть запускатися на певні події (наприклад, відкриття документа) і діставати доступ на модифікацію інших документів, відразу ж був використаний творцями вірусів зовсім не в благих цілях.

До подібних же прикладів слід віднести можливість запуску DLL-файлів з HLP-файлів. Здавалося б, відкривається нешкідливий текстовий файл довідки, а виявляється він може ініціювати виклик з DLL-бібліотеки без жодного повідомлення про це користувача.

5.2 Помилки, що приводять до можливості атак на інформацію

Двома основними категоріями помилок в розробці програмного забезпечення є:

- непрогнозована взаємодія суміжних програм, процесів, процедур і т.п. (інтерференція даних);
- невиконання набору угод, які розробник ПЗ вважав "само собою зрозумілим" (порушення неявних обмежень).

Інтерференція (тобто непередбачена взаємодія) даних між собою і даних з кодом є менш розповсюдженим, але небезпечнішим синдромом, ніж проблема обмежень за замовчуванням. Практично єдиним способом викликати нашарування даних одне на одне або даних на код програми, є спроба дійти при операції запису до межі області пам'яті, відведеної під даний блок інформації, і подолати її - тобто умисне переповнювання буфера. Це можливо тільки в тих ситуаціях, коли програмний код не проводить перевірки довжини значення, що записується. Для цілих і дробових чисел, значень часу і тому подібних типів даних, запис проводиться завжди у фіксованому об'ємі (2 байти, 4 байти, 10 байт). А ось для рядків і масивів даних перевірки довжини перед операціями запису необхідні. Для того, щоб примусити ЕОМ виконати код або записати дані туди, куди у нього немає прав запису, зловмисник спеціально примушує систему обробляти рядки дуже великої довжини або поміщати в масив

кількість елементів, більшу, ніж його об'єм. У разі успіху можливі або попадання частини рядка в сегмент коду або стека з подальшим виконанням, або модифікація будь-яких службових даних, що дозволить потім зловмисникові увійти у систему в обхід системи захисту. Вміст кінця рядка, що записується після переповнювання буфера в заборонену область пам'яті, підбирається спеціальним чином.

Проблема обмежень, які розробник програми вважає само собою зрозумілими, але які насправді можуть не виконуватися, зустрічається набагато частіше, але рідше приводить до будь-яких серйозних наслідків. Найчастіше результатом обробки подібних даних стає переривання роботи програми з повідомленням про помилку або просто "зависання". Тобто, даний клас атак використовується з метою проведення атаки "відмова в сервісі". Спектр можливих обмежень, не продуманих на етапі розробки ПЗ, надзвичайно широкий. Це можуть бути і негативний час або сума платежу, і текстове значення на місці очікуваного числа, і рядки, створені повністю з управляючих символів, і, звичайно ж, порожні рядки. Все це приводить до одного з головних правил розробки ПЗ : ретельно і повністю перевіряйте ті вхідні дані програми, які поступають в неї від людини або передаються по каналу зв'язку, незахищеному від модифікації.

5.3 Основні положення щодо розробки ПЗ

Наведемо узагальнені рекомендації щодо написання додатків, в яких вірогідність появи помилок як загальних, так і тих, що відносяться до інформаційної безпеки, суттєво зменшується. Дані рекомендації є необхідною, але не достатньою умовою інформаційної безпеки [37]:

- не використовуйте екзотичні і недокументовані можливості мови програмування: ви не упевнені в тому, як вони реалізуються насправді;
- оформляйте початковий текст ясно і чітко, використовуйте необхідні коментарі;
- використовуйте дужки для явного вказання порядку операцій: компілятор може оптимізувати виконання виразів і почати, скажімо, додавання F(1)+F(2)+F(3) з другого знаку "+", тим самим викликавши спочатку функцію F від 2, потім від 3, а тільки потім

від 1 – якщо у функції змінюються будь-які глобальні змінні це може призвести до непередбачуваних наслідків;
- при будь-якій слушній нагоді використовуйте передачу параметрів функції як аргументів, а не в глобальних змінних;
- використовуйте структурне програмування: розбивайте складні блоки коду на процедури з ясною структурою і легко контрольованим набором параметрів;
- ніколи не програмуйте недокументовані можливості: технологія "Reverse engineering" – дизасемблювання і обернена компіляція - на сьогоднішній день досягла величезних результатів, особливо відносно високорівневих мов програмування;
- закривайте файли відразу ж після закінчення роботи з ними, а якщо Ви записуєте важливу інформацію протягом довгого часу – періодично викликайте функції скидання файлового буфера на дисковий накопичувач;
- перевіряйте вільне місце на диску перед записом у файл: деякі операційні системи видають помилки при запису на переповнений диск нестандартним чином, результатом цього може бути втрата інформації;
- блокуйте файли і набори даних, якщо Ви звертаєтеся до них за записом з декількох паралельно працюючих процесів або програм;
- прагніть якомога більше скоротити час запису в спільно використовувані файли, а, отже, і час їх блокування;
- не будьте наперед упевненими, що програма запущена з тієї директорії, де розташований її виконавчий файл, - однією з перших команд після запуску програми явно змініть каталог на бажаний;
- при роботі із зовнішніми і мережевими пристроями, дисками будуйте цикли очікування так, щоб з них був можливий вихід після закінчення певного періоду очікування відповіді - тайм-ауту;
- дуже ретельно розробляйте схему синхронізації паралельних процесів, що працюють з одними і тими ж даними;
- ретельно перевіряйте алгоритми на синдром "мертвої петлі" - це ситуація, коли процес А, почавши змінювати об'єкт 1 і

заблокувавши його у зв'язку з цим, чекає зняття блокування з об'єкта 2, тоді як процес В, в той же самий час, почав змінювати об'єкт 2 і заблокувавши його, чекає зняття блокування з об'єкта 1; подібна проблема при такій схемі синхронізації теоретично нерозв'язна, єдиний вихід з неї – розглядати об'єкти 1 і 2 як єдине ціле з можливістю тільки сумісного блокування;

- акуратно виділяйте і звільняйте об'єкти в динамічній пам'яті;
- при необхідності використовуйте криптографію;
- ніколи не передавайте пароль відкритим текстом;
- використовуйте криптостійкі алгоритми шифрування і хешування;
- вичищайте блоки оперативної пам'яті після того, як інформація (паролі, ключі, конфіденційні дані), що знаходилася в них, стала непотрібною;
- завжди перевіряйте довжини рядків і масивів перед початком роботи з ними;
- вмонтовуйте у ваші системи вимогу реєстрації кожного оператора з унікальним паролем і записом як можна більшої кількості інформації про сеанс в лог-файл, недоступний для зміни оператором;
- ретельно тестуйте Ваші додатки, зокрема на великих і неправильних вхідних даних

5.4 Організація захисту від комп’ютерних вірусів

Що таке комп'ютерний вірус? Формальне означення цього поняття до цих пір не придумане, і є серйозні сумніви, що воно взагалі може бути.

Насамперед вірус – це шкідлива програма. Шкідливі програми (malicious software або malware) – це програми, які призначені для того, щоб чинити шкоду і використовувати ресурси комп’ютера, вибраного як мішень. Вони часто маскуються в легальних програмах або імітуються під них. В деяких випадках вони розповсюджуються самі по собі, переходячи по електронній пошті від одного комп’ютера до іншого, або через заражені файли і диски. Крім того, вірус - програма, що має здатність до самовідтворення. Така здатність є властивістю більшості типів вірусів.

Причому копії вірусу можуть взагалі не збігатися з оригіналом. Загальна класифікація шкідливих програм подана в [38].

Віруси можна розділити на класи за такими ознаками:

- за середовищем існування;
- за способом зараження;
- за особливостями використовуваних алгоритмів;
- за деструктивними можливостями.

За середовищем існування віруси поділяються на:

- файлові віруси;
- завантажувальні віруси;
- макровіруси;
- мережні віруси.

Файлові віруси діють одним з таких способів:

- впроваджуються в основному у виконувані файли, тобто у файли з розширеннями COM та EXE. Вони можуть впроваджуватись і у файли інших типів, але в такому випадку, як правило, вони ніколи не отримують управління, і, як наслідок, втрачають здатність до розмноження;
- створюють файли-двійники (компаньйон-віруси);
- використовують особливості організації файлової системи (Link-віруси).

Завантажувальні віруси діють такими методами:

- впроваджуються в завантажувальний сектор диска (надалі Boot-сектор або бут-сектор);
- впроваджуються в сектор, що містить програму завантаження системного диска (Master Boot Record – MBR);
- змінюють покажчик на активний boot-сектор.

Існують також файло-завантажувальні віруси, що заражають і файли, і завантажувальні сектори дисків.

Макровіруси заражають файли-документи і електронні таблиці відомих програмних продуктів.

Мережеві віруси використовують для свого розповсюдження протоколи або команди комп’ютерних мереж та електронної пошти.

Віруси за способом зараження поділяються так:

- резидентні віруси при зараженні (інфікуванні) комп’ютера залишають в оперативній пам’яті свою резидентну частину,

яка потім перехоплює звернення операційної системи до об’єктів зараження (файлів, завантажувальних секторів тощо) і впроваджується в них.

- нерезидентні віруси не заражають пам’ять комп’ютера і є активними лише обмежений проміжок часу. Деякі віруси залишають в пам’яті невеликі резидентні програми, які не розповсюджують віруси. Такі віруси вважаються нерезидентними.

Віруси за особливостями використовуваних алгоритмів такі:

- прості віруси – це віруси-паразити, вони змінюють вміст файлів і секторів дисків і можуть бути досить легко виявлені та знешкоджені;
- стелс-віруси (або віруси-невидимки) – це віруси, які повністю або частково приховують себе в системі. Найбільш розповсюдженим стелс-алгоритмом є перехоплення запитів ОС на читання-записування заражених об’єктів;
- віруси-мутанти, які можуть використовувати алгоритми шифрування-розшифрування та поліморфізму, завдяки яким копії одного і того ж вірусу не мають жодного ланцюжка байтів, що повторюються. Поліморфні віруси досить важко виявити, вони не мають сигнатури, тобто не містять жодної сталої ділянки коду

Віруси за деструктивними можливостями можна подати у такому розрізі:

- нешкідливі віруси, які ніяким чином не впливають на роботу комп’ютера, крім зменшення вільної пам’яті на диску в результаті свого поширення;
- безпечні віруси, вплив яких обмежується зменшенням вільної пам’яті на диску і графічними, звуковими та іншими ефектами;
- небезпечні віруси, робота яких може призвести до серйозних збійних ситуацій у роботі комп’ютера;
- дуже небезпечні віруси, активність яких може призводити до втрати програми, знищення даних, стирання необхідної для роботи комп’ютера інформації, яка записана в системних областях пам’яті, і навіть сприяти прискореному зносу рухомих частин механізмів, наприклад, головок вінчестера.

Насамперед необхідно відзначити, що захистити комп'ютер від вірусів може лише сам користувач. Основні способи захисту такі:

- систематичне архівування інформації;
- обмеження ненадійних контактів;
- своєчасне застосування антивірусних засобів може захистити комп'ютер від зараження або забезпечити мінімальний збиток, якщо зараження все-таки відбулося.

Єдиним стовідсотковим за надійністю методом захисту від втрати важливої інформації є її резервне копіювання на захищені від записування пристрої зберігання даних. Більше того, архівуванням також не можна нехтувати, оскільки втратити інформацію можна не лише через віруси, але й через стрибки напруги в мережі, поломки обладнання й т.д. Жодна антивірусна програма не зрівняється за надійністю з архівуванням інформації. Справа в тому, що на будь-який алгоритм антивірусу завжди найдеться алгоритм вірусу, невидимого для цього антивірусу.

Друге правило, яке частково гарантує збереження інформації, – це обмеження копіювання даних з ненадійних джерел. Як би ми не старалися, обмін інформацією з іншими користувачами і робота в локальних або глобальних мережах неминучі. Однак деякі правила для себе все-таки виділити можна. По-перше, необхідно намагатися не запускати неперевірені файли, у тому числі отримані з комп'ютерної мережі. Бажано використовувати тільки програми, отримані з надійних джерел. Перед запуском нових програм обов'язково варто перевірити їх одним або декількома антивірусами. По-друге, варто обов'язково користуватися тільки тими джерелами та іншими файлами, які добре зарекомендували себе, хоча це не завжди рятує (наприклад, на WWW-сервері Microsoft досить довгий час перебував документ, заражений макровірусом “Wazzu”). По-третє, необхідно обмежити коло людей, які допущені до роботи на конкретному комп'ютері. Практика показує, що найбільш уразливі комп'ютери – багатокористувацькі. І нарешті, відповідно до четвертого правила, варто купувати тільки дистрибутивне програмне забезпечення в офіційних продавців. Безкоштовні, умовно безкоштовні або піратські копії можуть призвести до зараження.

При існуючому різноманітті вірусів і їх мутацій запобігти зараженню може тільки повнофункціональна антивірусна система, що має в своєму арсеналі всі відомі технології боротьби з «інфекційними хворобами»: не

тільки сканер-поліфаг, але і резидентний on-line-монітор, засоби контролю програмної цілісності і евристичного пошуку вірусних сигнатур. Всі основні фірми-постачальники антивірусного забезпечення регулярно і досить часто оновлюють файли сигнатур вірусів, а при появі особливо шкідливого вірусу створюють додатковий екстрений випуск. Ще зовсім недавно вважалося, що сигнатури потрібно оновлювати щомісячно, але в нашу епоху нових вірусів, можливо, буде розумним перевіряти їх щотижня вручну або за допомогою автоматичного оновлення антивірусної програми. Однак на всі 100% захиститися від вірусів практично неможливо. Більш детально ці питання розглянуто в [38].

Контрольні питання

1. Які основні компоненти включає програмне забезпечення?
2. Які методи захисту інформації від несанкціонованого доступу застосовуються в ОС?
3. Який основний недолік матриць та списків доступу і паролів?
4. Охарактеризуйте політику безпеки в популярних ОС.
5. Помилки в якому програмному забезпеченні є основним шляхом проникнення зловмисника як на сервери, так і на робочі станції?
6. Які основні категорії помилок в розробці програмного забезпечення?
7. Наведіть узагальнені рекомендації щодо написання додатків, в яких вірогідність появи помилок як загальних, так і тих, що відносяться до інформаційної безпеки, суттєво зменшується.
8. За якими ознаками віруси поділяються на класи?
9. Наведіть класифікацію вірусів середовищем існування.
10. Наведіть класифікацію вірусів за способом зараження.
11. Наведіть класифікацію вірусів за особливостями алгоритмів, що ними використовуються.
12. Дайте класифікацію вірусів за деструктивними можливостями.
13. Охарактеризуйте файлові віруси.
14. Охарактеризуйте завантажувальні віруси
15. Наведіть основні способи боротьби з вірусами.

ВИСНОВКИ

Знання основ теорії інформації є обов'язковим для інженера будь-якої спеціальності, пов'язаної з інформаційними технологіями. В навчальному посібнику охоплено основи всіх напрямків теорії інформації, які вже давно виділились в окремі дисципліни. Це надає можливість студентам краще зрозуміти взаємозв'язки між окремими напрямками як частинами єдиного цілого.

У зв'язку із зростанням ролі комп'ютерних мереж у повсякденному житті людей, інтерес до теорії інформації та її невід'ємних складових кодування і криптології знову збільшується. Причому, широке застосування знаходять однаковою мірою три основні складові теорії інформації, які мають прикладний характер: економне кодування, завадостійке кодування та криптологія. Методи теорії інформації, які раніше були лише предметом теоретичних досліджень, стали основою промислових стандартів, що використовуються в мережах. Не в останню чергу це викликано тим, що через мережі передаються повідомлення не тільки розважального характеру, але і важливі як з точки зору окремої людини, так і для суспільства в цілому, в тому числі і секретні. Пошкодження або попадання в руки зловмисників таких документів може призвести до непередбачуваних наслідків.

Про зростання інтересу до теорії інформації та її складових свідчить поява в останні роки сайтів в Інтернет, які спеціалізуються на розповсюдженні безкоштовної навчальної інформації про методи теорії інформації. Можна відзначити наприклад сайт www.compression.ru, який дає всебічну інформації про методи ущільнення даних та про сучасний стан досліджень в даній області. Основним автором даного сайту є Ватолін Д., там же розміщені і глави його підручника [10]. Є також ряд сайтів, які дозволяють безкоштовно для некомерційного використання скачувати відсканові посібники, підручники, монографії з теорії інформації як ті, що давно видавались і їх практично неможливо знайти в паперовому вигляді, так і ті, що тільки вийшли з друку. Деякі з них такі: www.mirknig.com, www.knigka.info, www.ihtik.lib.ru, http://lib.prometey.org, www.litportal.kiev.ua, http://physicsbooks.narod.ru.

ЛІТЕРАТУРА

1. Шеннон К. Работы по теории информации и кибернетике./Пер. с англ. под ред. Р. Л. Добрушина и О. Б. Лупанова. – М.: Иностранная литература, 1963. – 106 с.
2. Дмитриев В. И. Прикладная теория информации. – М.: Высш. шк., 1989. – 420 с.
3. Основи теории информации и кодирования/И. В. Кузьмин, В. А. Кедрус. – К: Вища шк., 1986.-238 с.
4. Хэмминг Р. В. Теория кодирования и теория информации. Пер. с англ. – М.: Радио и связь, 1983. – 176 с.
5. Харкевич А. А. Борьба с помехами. – М.: Наука, 1965. – 270 с.
6. Колмогоров А. Н. Три подхода к определению понятия "количество информации"// Проблемы передачи информации. – 1965 - №1. - С. 3-11.
7. Зюко А. Г. Элементы теории передачи информации. – К.: Техніка, 1969. – 300 с.
8. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки: Пер. с англ.. - М.: Мир, 1976. – 589 с., ил.
9. Основы теории передачи информации. Ч. 1. Экономное кодирование/ В. И. Шульгин. – Учебное пособие. – Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2003. – 102 с.
10. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. – 384 с.
11. Лужецький В. А. Високонадійні математичні Фібоначчі-процесори: Монографія. Вінниця: ВДТУ, 2000. – 248 с.
12. Майданюк В. П. Методи і засоби комп’ютерних інформаційних технологій. Кодування зображень. Навчальний посібник. – Вінниця: ВДТУ, 2001. – 63 с.
13. Балашов К. Ю. Сжатие информации: анализ методов и подходов. – Минск: 2000. – 42 с (Препринт / Ин-т техн. Кибернетики НАН Беларуси; № 6)
14. Семенюк В. В. Экономное кодирование дискретной информации. – СПб.: СПбГИТМО (ТУ), 2001. – 115 с.

15. Кричевский Р. Е. Сжатие и поиск информации. - М.: Радио и связь, 1989. - 168 с.
16. Фомин А. А. Основы сжатия информации. - СПб.: СПбГТУ, 1998. – 85 с.
17. Гонсалес Р., Вудс Р. Цифровая обработка изображений.- М.: Техносфера, 2005. – 1072 с.
18. Сэломон Д. Сжатие данных, изображений и звука. – М.: Техносфера, 2004. – 368 с.
19. Вернер М. Основы кодирования. Учебник для ВУЗов. - М: Техносфера, 2004. – 288 с.
20. Золотарёв В. В., Овечкин Г. В. Помехоустойчивое кодирование. Методы и алгоритмы: Справочник / Под. ред. чл.-кор. РАН Ю. Б. Зубарева. - М.: Горячая линия-Телеком, 2004. - 126 с.
21. Блейхут Р. Теория и практика кодов контролирующих ошибки: Пер. с англ. – М.: Мир, 1986. – 576 с.
22. Касами Т., Токура Н, Ивадари Е., Инагаки Я. Теория кодирования: Пер. с японск. - М.: Мир, 1978. – 576 с.
23. Прокис Дж. Цифровая связь. Пер. с англ. /Под ред. Д. Д. Кловского. - М.: Радио и связь, 2000. - 800 с.
24. Скляр Б. Цифровая связь и практическое применение: Пер. с англ. – М.: Издательський дом «Вильямс», 2003. – 1104 с.
25. Основы теории передачи информации. Ч. 2. Помехоустойчивое кодирование/ В. И. Шульгин. – Учебное пособие. – Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2003. – 87 с.
26. Кларк Дж., Кейн Дж. Кодирование с исправлением ошибок в системах цифровой связи. Пер. с англ. – М.: Радио и связь, 1987. – 392 с.
27. Банкет В. Л. Сверточные коды в системах передачи информации: Учеб. пособие / Одесск. электротехн. ин-т связи им. А. С. Попова. Одесса, 1986. – 57 с.
28. Секреты и ложь. Безопасность данных в цифровом мире/ Б. Шнайдер. – СПб.: Питер, 2003. – 386 с.
29. Основы информационной безопасности. Учебное пособие для вузов / Е. Б. Белов, В. П. Лось, Р. В. Мещеряков, А. А. Шелупанов. -М.: Горячая линия - Телеком, 2006. - 544 с.

30. Хорошко В. О., Азаров О. Д., Шелест Ю. С., Яремчук Ю. Є. Основи комп'ютерної стеганографії. Навчальний посібник. – Вінниця: ВДТУ, 2003. – 143 с.
31. Брассар Ж. Современная криптология: Пер. с англ. - М.: Издательско-полиграфическая фирма ПОЛИМЕД, 1999.-176 с.
32. Рябко Б. Я., Фионов А.Н. Криптографические методы защиты информации: Учебное пособие для вузов. - М.: Горячая линия - Телеком, 2005. – 229 с.
33. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритмы криптографического преобразования.
34. Методичні вказівки до виконання лабораторних робіт з дисципліни "Операційні системи ЕОМ"/Укладач Майданюк В. П., Романюк О. Н. – Вінниця: ВДТУ, 1997. – 41 с.
35. Соколов А. В., Степанюк О. М. Защита от компьютерного терроризма. – СПб.: БХВ-Петербург, 2002. – 496 с.
36. Казарин О. В. Безопасность программного обеспечения компьютерных систем. - М.: МГУЛ, 2003.
37. Беляев А. Методы и средства защиты информации. Курс лекций в Череповецком филиале СПбГТУ. - http://science.metacom.ru
38. Каплун В. А., Майданюк В. П. Захист операційних систем. Навчальний посібник. - Вінниця: ВНТУ, 2007. – 185 с.

СЛОВНИК ОСНОВНИХ ТЕРМІНІВ (GLOSSARY)

Алгоритм - algorithm
Апостеріорний - aposterior
Апріорний - aprior
Відкритий ключ - public key
Двійкова одиниця - binary digit(bit)
Двійковий симетричний канал - binary symmetric channel
Двійкові коди - binary codes
Декодер - decoder
Дешифрування - decryption
Джерело без пам’яті - memoryless source
Джерело з пам’яттю - source with memory
Джерело повідомлень - source of messages
Дискретний – discrete
Ентропія – entropy
Ергодичність – ergodic
Завада – noise
Закритий ключ - private key
Зв’язок - communication
Знак – character
Імовірність – probability
Канал без завад - noiseless channel
Канал з завадами - channel with noise
Канали без пам’яті - memoryless channel
Канали з пам’яттю - channels with memory
Кібернетика – cybernetics
Кодек – codec
Кодер – coder
Кодування - coding, encoding
Кодування без втрат – lossless coding
Кодування з втратами – lossy coding
Комбінаторний – combinatory
Корегуючі коди - correctings codes
Криптографія – cryptography
Криптологія – criptologic
Ланцюг Маркова - Markov chain
Миттєві коди - instantaneous code
Модель - model
Надмірність – redundancy
Неперервний - continuous
Оптимальне кодування - optimal coding
Пам’ять - memory
Парність - parity
Префіксні коди - prefix codes
Програмне забезпечення - software
Продуктивність джерела - source productivity
Пропускна здатність – capacity
Сигнал – signal
Синтез – synthesis
Стан – state
Стаціонарний – stationary
Теорія інформації - information theory
Швидкість маніпуляції - manipulation speed
Швидкість передачі інформації - information rate
Шифрування - encryption

Навчальне видання

Володимир Павлович Майданюк

Кодування та захист інформації

Навчальний посібник

Оригінал-макет підготовлено автором

Редактор В. О. Дружиніна
Коректор З. В. Поліщук

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку
Формат 29,7х42 1/4
Друк різографічний
Тираж прим.
Зам. №

Гарнітура Times New Roman
Папір офсетний
Ум. друк. арк.

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ