Дискретна математика - 2

Змістовий модуль 6. Теорія кодів
Тема 4. Оптимальне кодування (закінчення). Код Гаффмана. Стиснення даних. Коди, стійкі до завад.

План лекції

- **Оптимальне кодування. Код Гаффмана – оптимальний код**
- **Стиснення даних. Алгоритм Лемпеля – Зіва**
- **Коди, стійкі до завад**
- **Виявлення однієї помилки**
- **Виправлення однієї помилки – один приклад коду Геммінга**

Ефективність методу Фано полягає ось у чому. Що імовірнішою є поява букви, то швидше вона утворить „самостійну” підмножину, і тому її буде закодовано коротшим елементарним кодом. Тому метод Фано доволі ефективний. Але чи завжди він дає змогу отримати оптимальний код? Виявляється, що ні.

Код Гаффмана – оптимальний код

Поданий далі метод побудови оптимального коду потребує тонших міркувань ніж ті, що були використані для побудови коду Фано. Передусім сформулюємо й доведемо деякі допоміжні твердження.

Лема 1. В оптимальному коді букву з меншою ймовірністю її появи не може бути закодовано коротшим словом. Інакше кажучи, для оптимального коду з того, що $p_i < p_j$, випливає, що $l_i \geq l_j$.

Доведення. Припустимо протилежне: нехай є дві букви a_i та a_j такі, що $p_i < p_j$ та $l_i < l_j$. Поміняємо місцями β_i та β_j у схемі кодування. Тоді, середня довжина елементарних кодів зміниться на

$$p_i l_j + p_j l_i - p_i l_i - p_j l_j = (p_i - p_j) l_j - (p_i - p_j) l_i = (p_i - p_j)(l_j - l_i) < 0,$$

тобто зменшиться, що суперечить оптимальності коду. Лему доведено.

Лема 2. Якщо код оптимальний, то можна так перенумерувати букви алфавіту $A=\{a_1, \ldots, a_r\}$ і відповідні елементарні коди $\beta_1, \beta_2, \ldots, \beta_r$, що $p_1 \geq p_2 \geq \ldots \geq p_r$, і при цьому $l_1 \leq l_2 \leq \ldots \leq l_r$.

Доведення. Якщо $p_i > p_{i+1}$, то з леми 6.1 випливає, що $l_i \leq l_{i+1}$. Якщо ж $p_i = p_{i+1}$, але $l_i > l_{i+1}$, то поміняємо місцями нумерацію букв a_i, a_{i+1} і відповідних елементарних кодів. Повторюючи цю процедуру потрібну кількість разів, одержимо бажану нумерацію. Лему доведено.

З нерівності $l_1 \leq l_2 \leq \ldots \leq l_r$ випливає, що букву a_r (найменш імовірну) закодовано словом β_r із найбільшою довжиною.

Лема 3. В оптимальному двійковому коді завжди знайдеться принаймні два елементарних коди найбільшої довжини, яка дорівнює l_r, і таких, що вони відрізняються один від одного лише в останньому символі.

Доведення. Припустимо, що це не так. Тоді можна відкинути останній символ елементарного коду β_r, не порушуючи властивості префіксності. При цьому, очевидно, зменшиться середня довжина елементарного коду. Це суперечить твердженню, що код оптимальний.

Теорема 1. Існує такий оптимальний код, у якому елементарні коди двох найменш імовірних букв a_{r-1} і a_r відрізняються лише останнім символом.

Доведення. За лемою 3 знайдеться елементарний код β_t, який має ту саму довжину, що й β_r, і відрізняється від нього лише останнім символом. Із лем 1 та 2 випливає, що $l_t = l_{t+1} = \ldots = l_r$. Якщо виявиться, що $t \neq r-1$, то поміняємо місцями β_t та β_{r-1}, очевидно, що нерівність $l_1 \leq l_2 \leq \ldots \leq l_r$ при цьому не порушиться.

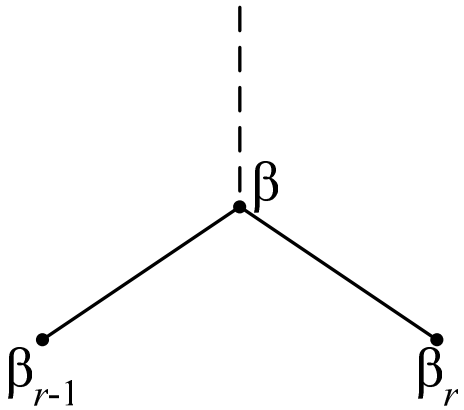


Рис. 1

Теорема 1 дає змогу розглядати лише такі схеми алфавітного кодування, у яких елементарні коди β_{r-1} та β_r (для двох найменш імовірних букв a_{r-1} і a_r) мають найбільшу довжину й відрізняються тільки останнім символом. Це означає, що листки β_{r-1} та β_r кодового дерева оптимального коду мають бути з'єднані в одній вершині попереднього рівня (рис. 1).

Розглянемо новий алфавіт $A_1 = \{a_1, \ldots, a_{r-2}, a\}$ із розподілом ймовірностей $P_1 = (p_1, \ldots, p_{r-2}, p)$, де $p = p_{r-1} + p_r$. Його одержано з алфавіту

$$A = \{a_1, \ldots, a_{r-2}, a_{r-1}, a_r\}$$

об'єднанням двох найменш імовірних букв a_{r-1} та a_r в одну букву a з імовірністю $p = p_{r-1} + p_r$. Говорять, що A_1 отримано *стисненням* алфавіту A.

Нехай для алфавіту A_1 побудовано схему σ_1 з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta$. Інакше кажучи, є якесь кодове дерево з листками $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta$. Схемі σ_1 можна поставити у відповідність схему σ з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta_{r-1}, \beta_r$ для початкового алфавіту A, узявши $\beta_{r-1} = \beta 0$, $\beta_r = \beta 1$ (тобто елементарні коди β_{r-1} та β_r одержують з елементарного коду β приписуванням справа відповідно 0 та 1).

Процедуру переходу від схеми кодування σ_1 до схеми σ називають *розщепленням*.

Теорема 2. Якщо схема кодування σ_1 є оптимальною для алфавіту A_1, то схема σ є оптимальною для алфавіту A.

Доведення. Із способу побудови схеми кодування σ можна дійти висновку, що середні довжини $l_{\text{сер}}^{\sigma}$ та $l_{\text{сер}}^{\sigma_1}$ елементарних кодів у схемах σ та σ_1 пов'язані співвідношенням $l_{\text{сер}}^{\sigma} = l_{\text{сер}}^{\sigma_1} + p$.

Припустимо, що схема σ не оптимальна для алфавіту A, тобто існує схема Σ, для якої $l_{\text{сер}}^{\Sigma} < l_{\text{сер}}^{\sigma}$. Нехай її елементарні коди – β'_1, $\beta'_2,\ldots,\beta'_{r-2}$, β'_{r-1}, β'_r. Можна вважати, що листки β'_{r-1} та β'_r кодового дерева схеми Σ відповідають двом найменш імовірним буквам алфавіту A і відрізняються лише останнім символом.

Розглянемо схему Σ_1 для алфавіту A_1. Елементарні коди схеми Σ_1 – β'_1, $\beta'_2,\ldots,\beta'_{r-2}$, β'. Елементарний код β' одержують відкиданням останнього символу від β'_r (або від β'_{r-1}, результат буде тим самим). Тому середні довжини кодів $l_{\text{сер}}^{\Sigma}$ та $l_{\text{сер}}^{\Sigma_1}$ пов'язані співвідношенням $l_{\text{сер}}^{\Sigma} = l_{\text{сер}}^{\Sigma_1} + p$.

Зі співвідношень $l_{\text{сер}}^{\sigma} = l_{\text{сер}}^{\sigma_1} + p$, $l_{\text{сер}}^{\Sigma} = l_{\text{сер}}^{\Sigma_1} + p$ та $l_{\text{сер}}^{\Sigma} < l_{\text{сер}}^{\sigma}$ випливає, що $l_{\text{сер}}^{\Sigma_1} < l_{\text{сер}}^{\sigma_1}$. Це суперечить оптимальності схеми σ_1 для алфавіту A_1. Теорему доведено.

З теореми 2 випливає такий метод побудови оптимальної схеми алфавітного кодування. Спочатку послідовно стискають алфавіт A до отримання алфавіту з двох букв, оптимальна схема кодування для якого очевидна: першу букву кодують символом 0, другу букву – символом 1. Потім послідовно розщеплюють одержану схему. Очевидно, що отримана в результаті схема є префіксною.

Описаний метод кодування запропоновано 1952 року американським вченим в галузі електротехніки і електроніки Д. Гаффманом (D. Huffman).

Приклад. Нехай задано розподіл ймовірностей P = (0.4, 0.15, 0.15, 0.15, 0.15). Побудуємо код методом Гаффмана. Розв'язок наведено в наступній таблиці.

Таблиця реалізації методу Гаффмана для побудови оптимального коду

Букви алфавіту A	Ймовірності та позначення елементарних кодів							
	Початковий алфавіт A		Стиснуті алфавіти					
			A_1		A_2		A_3	
a_1	0.4	1	0.4	1	0.4	1	0.6	0
a_2	0.15	010	0.3	00	0.3	00	0.4	1
a_3	0.15	011	0.15	010	0.3	01		
a_4	0.15	000	0.15	011				
a_5	0.15	001						

Кожний з алфавітів A_1, A_2, A_3 одержують стисненням попереднього алфавіту. Алфавіт A_3 складається з двох букв, тому оптимальна схема містить лише два елементарні коди – символи 0 і 1. Послідовне розщеплення дає оптимальну схему для початкового алфавіту A (у процесі розщеплення потрібно рухатися проти стрілок).

Середня довжина побудованого коду, яка дорівнює $l_{\text{сер}}^{H} = 0.4 \times 1 + 4 \times 0.15 \times 3 = 2.2$, як це випливає з попереднього, мінімально можлива для даного розподілу ймовірностей P. Тут індекс H означає, що код отримано методом Гаффмана.

У попередній лекції цей же приклад розв'язано методом Фано і одержано код із середньою довжиною $l_{\text{сер}}^{F} = 2.3$. Отже, метод кодування Фано не завжди дає оптимальний код.

За допомогою алгоритму Гаффмана можна безпосередньо побудувати кодове дерево оптимального коду, яке називають *деревом Гаффмана*. Бінарне дерево, що відповідає оптимальному коду, будують знизу вверх, починаючи з $|A| = r$ листків, і виконують $r - 1$ злиття. *Злиття* полягає в побудові нового дерева із двох наявних дерев (або листків) із найменшими ймовірностями. При цьому листок або дерево з більшою ймовірністю завжди утворює ліве піддерево, а сума ймовірностей лівого й правого піддерев дорівнює ймовірності отриманого дерева (її записують у корінь). Поміщають 0 на ребро до лівого піддерева, 1 – на ребро до правого піддерева. Перед кожним злиттям імовірності упорядковують за спаданням.

Зауваження. Алгоритм Гаффмана належить до жадібних алгоритмів.

Приклад. Побудуємо дерево Гаффмана для розподілу ймовірностей P = (0.34, 0.18, 0.17, 0.16, 0.15). Оскільки є всього п'ять букв, то для побудови дерева потрібно зробити чотири злиття. На кожному кроці зливають два піддерева з найменшими ймовірностями. Одержують нове дерево із сумарною ймовірністю лівого та правого піддерев і впорядковують ймовірності за спаданням. Листки зображають прямокутниками, у яких букви та їхні ймовірності відокремлюють двокрапкою (рис. 2). Внутрішні вершини зображають кружечками, у яких зазначають суми ймовірностей їхніх синів. Динаміку роботи алгоритму відображено на рис. 3, *а – г*.

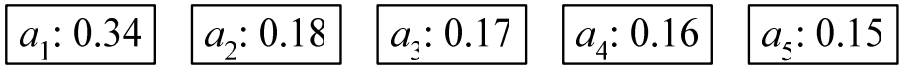
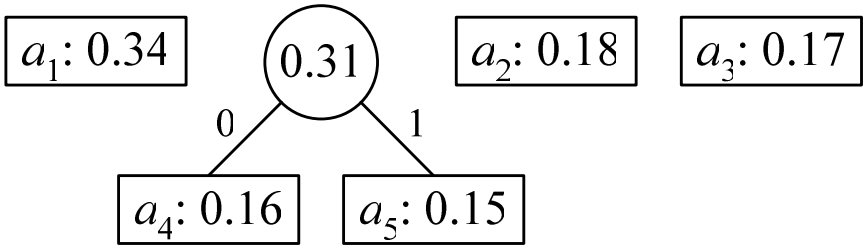
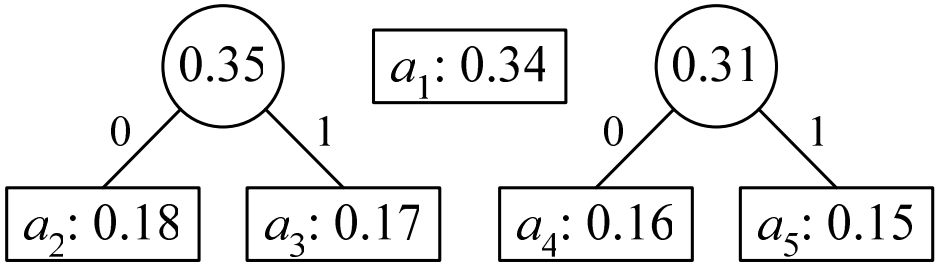

a_1: 0.34
a_2: 0.18
a_3: 0.17
a_4: 0.16
a_5: 0.15

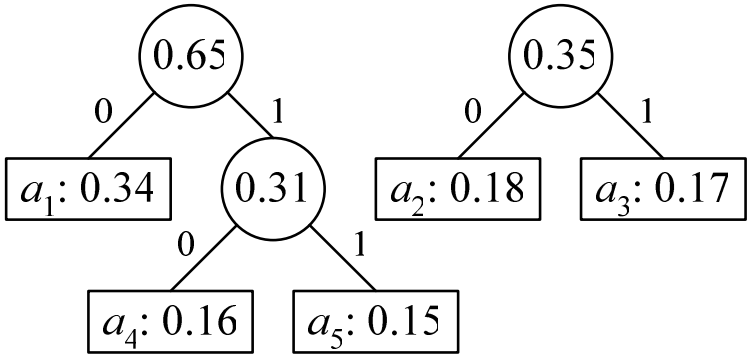
Рис. 2


a_1: 0.34
0.31
a_2: 0.18
a_3: 0.17
0
1
a_4: 0.16
a_5: 0.15

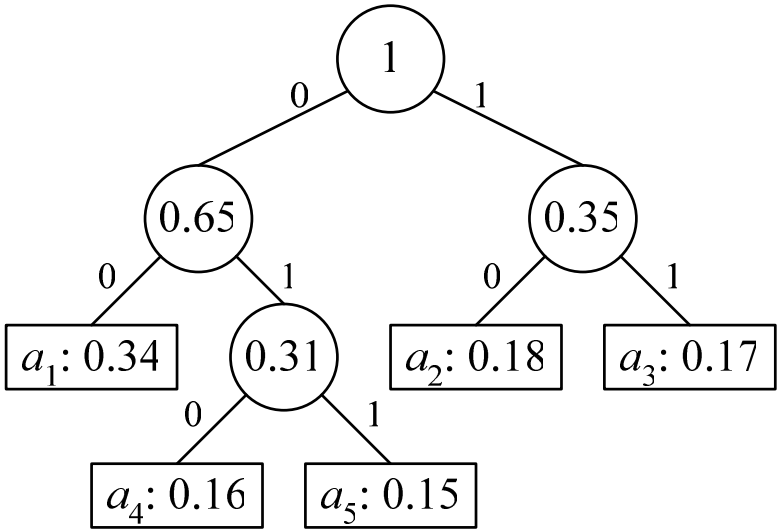
а


0.35
a_1: 0.34
0.31
0
1
0
1
a_2: 0.18
a_3: 0.17
a_4: 0.16
a_5: 0.15

б


0.65
0.35
0
1
0
1
a_1: 0.34
0.31
a_2: 0.18
a_3: 0.17
0
1
a_4: 0.16
a_5: 0.15

в


1
0
1
0.65
0.35
0
1
0
1
a_1: 0.34
0.31
a_2: 0.18
a_3: 0.17
0
1
a_4: 0.16
a_5: 0.15

г

Рис. 3

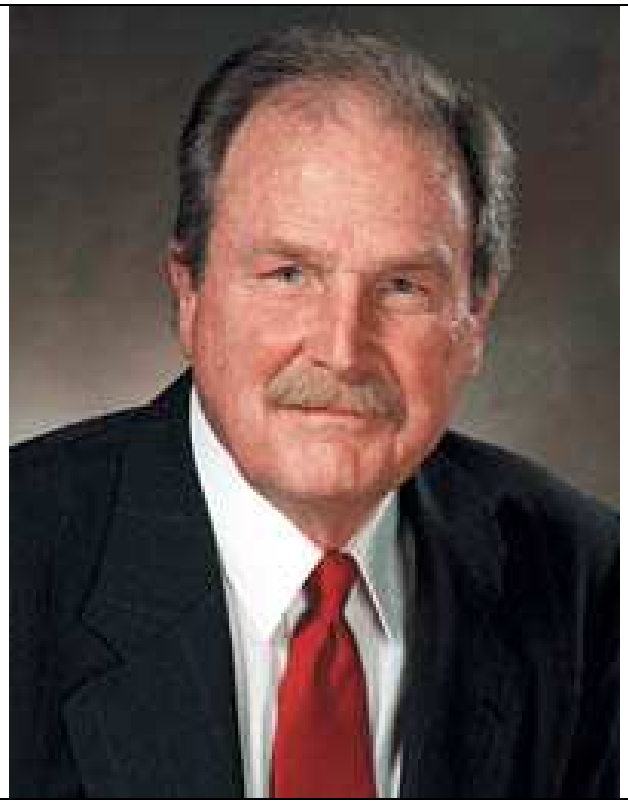
Щоб отримати елементарний код для певної букви, потрібно пройти єдиний шлях від кореня до відповідного листка, записуючи послідовність бітів. Одержимо таку оптимальну схему алфавітного кодування за заданого розподілу ймовірностей:

$$a_1 \to 00,\ a_2 \to 10,\ a_3 \to 11,\ a_4 \to 010,\ a_5 \to 011.$$

Узагалі кажучи, оптимальна схема не єдина. Так, у наступній таблиці для розподілу ймовірностей $P = (0.4,\ 0.2,\ 0.2,\ 0.1,\ 0.1)$ наведено три різні оптимальні схеми ($l^1_{\text{сер}} = l^2_{\text{сер}} = l^3_{\text{сер}} = l_* = 2.2$).

Таблиця для трьох різних оптимальних схем

Букви алфавіту	Ймовірності	Схема 1	Схема 2	Схема 3
a_1	0.4	11	1	1
a_2	0.2	10	011	01
a_3	0.2	01	010	001
a_4	0.1	001	001	0001
a_5	0.1	000	000	0000



Девід Гаффман (Хаффман) народився в 1925 р. у в штаті Огайо, США. Отримав ступінь бакалавра електротехніки в державному університеті Огайо у віці 18 років. Потім він служив в армії офіцером підтримки радара на есмінці, який допомагав знешкоджувати міни в японських і китайських водах після Другої світової війни. Згодом отримав ступінь магістра в університеті Огайо й ступінь доктора в Массачусетському інституті технологій (MIT). Хоча Гаффман більше відомий за розробку методу побудови оптимального коду, він так само зробив важливий внесок у безліч інших областей (по більшій частині в електроніці). Він довгий час очолював кафедру комп'ютерних наук в MIT.

У 1974 році, будучи вже заслуженим професором, він подав у відставку. Гаффман отримав ряд цінних нагород.

Помер 1999 року у віці 74 років.

Стиснення даних. Алгоритм Лемпеля – Зіва

Припустимо, що деяке повідомлення, наприклад текст, закодовано так, що кожний символ займає один байт (вісім бітів). Зазначимо, що рівномірне кодування неоптимальне для текстів. Справді, у текстах зазвичай використовується істотно менше ніж 256 символів – приблизно 60 – 80 із урахуванням знаків пунктуації, цифр, великих і малих букв. Окрім того, ймовірності появи букв різні, і для кожної природної мови відомі частоти появи букв у тексті. Отже, для алфавіту природної мови за допомогою алгоритму Гаффмана можна побудувати оптимальне алфавітне кодування текстів.

Методи кодування, які дають змогу побудувати без утрати інформації коди повідомлень з меншою довжиною ніж початкові повідомлення, називають методами *стиснення* (або *пакування*) даних. Якість стиснення визначають *коефіцієнтом стиснення*, який зазвичай вимірюють у відсотках. Цей коефіцієнт показує, на скільки відсотків закодоване повідомлення коротше від початкового. У разі стиснення текстових файлів спеціалізованими програмами-архіваторами коефіцієнт досягає 70% і більше, що набагато краще, ніж може дати оптимальний алфавітний код.

Це означає, що в таких програмах використовують **не** алфавітне кодування.

Розглянемо спосіб кодування, який полягає в *попередній побудові словника*.

1. Початкове повідомлення за певним алгоритмом розбивається на послідовності символів, які називають словами (слово може декілька разів входити в текст повідомлення).

2. Одержану множину слів уважають буквами нового алфавіту. Для цього алфавіту будують роздільну схему кодування (наприклад, рівномірного кодування або оптимального алфавітного кодування, якщо для кожного слова пораховано кількість його входжень у текст). Одержану схему називають *словником*, бо вона зіставляє слову його код.

3. Далі код повідомлення будують як пару – код словника та послідовність кодів слів із даного словника.

4. Під час декодування початкове повідомлення відновлюється за допомогою заміни кодів слів на слова зі словника.

Зауваження. На кроці 2 як раз і застосовують алгоритм Гаффмана для побудови оптимального коду.

На практиці використовують наступну ідею, відому як *адаптивне стиснення*. За один прохід по тексту одночасно динамічно будується словник і кодується текст. При цьому словник не зберігається – під час декодування використовується той самий алгоритм побудови словника, і тому словник динамічно відновлюється.

Практична реалізація цієї ідеї відома як *алгоритм Лемпеля – Зіва*. Спочатку словник D : **array [int] of string** містить порожнє слово, що має код 0. Далі в тексті послідовно виокремлюються слова. Слово, яке виокремлюється, – це найдовше слово з тих, що вже наявні в словнику, плюс іще один символ. У стиснене подання записується знайдений код слова та буква розширення, а словник доповнюється розширеною комбінацією.

Коди, стійкі до завад

Надійність інформації першочергово залежить від надійності носія інформації. Це так звана апаратна надійність, її досягають технічними засобами. У теорії кодування вивчають математичні методи підвищення надійності інформації, тобто підвищення надійності інформації через її *раціональне подання*.

Питання завадостійкості розглядають у двох формулюваннях, відповідно до класифікації можливих завад. Перший тип – заміщення символів у деяких позиціях кодових комбінацій. Їх називають *адитивними помилками*, бо результат їхньої дії математично можна подати як порозрядне додавання якогось вектора-помилки до кодової комбінації.

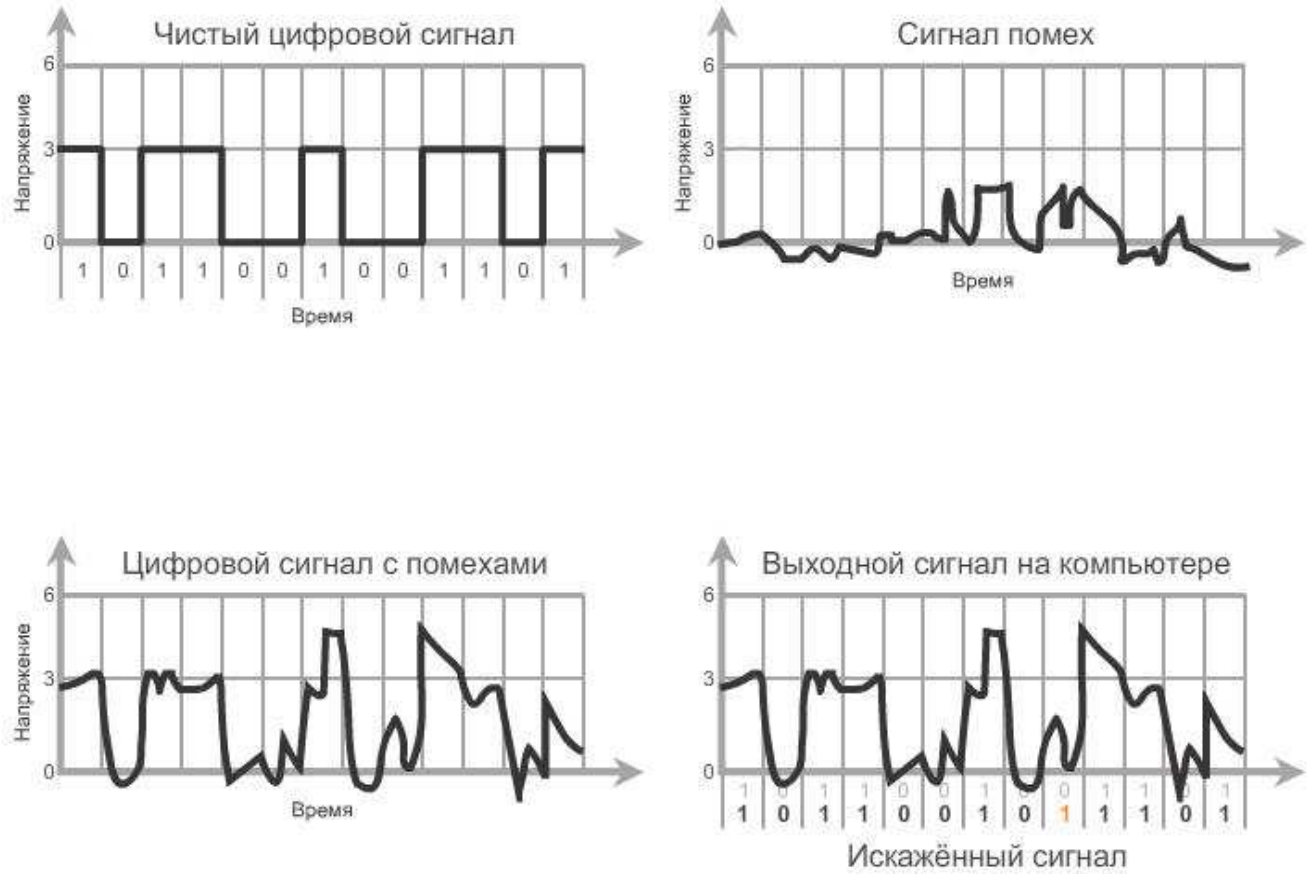


Рис. 4

На рис. 4 зображено чистий цифровий сигнал, сигнал завад, цифровий сигнал з завадами, вихідний сигнал на комп'ютері. Як можна побачити, передавалося повідомлення 1011001001101, а отримано повідомлення 1011001011101; тут вектор-помилка 0000000010000.

Другий тип завад – *синхронізаційні завади*, результатом котрих є неузгодженість схеми кодування й схеми декодування.

Засобом підвищення надійності подання інформації в кожному випадку є введення в записи певної *надлишковості*.

Далі розглядатимемо лише перший тип завад, тобто адитивні помилки. Головними формами завадостійкості каналу зв'язку є здатність *виявляти якусь кількість помилок*, або сильніша здатність – виправляти помилки. Ці властивості і методи розглядатимемо на *моделі двійкового рівномірного кодування*.

У цій лекції розглянемо один частковий випадок рівномірного двійкового кодування, а саме, уважатимемо, що не тільки алфавіт $B=\{0, 1\}$, але й алфавіт повідомлень $A=\{0, 1\}$. Розглянемо схему рівномірного кодування $\sigma_{k,n}$ із параметрами k, n:

$$\alpha_1 \to \beta_1,$$
$$\alpha_2 \to \beta_2,$$
$$\alpha_3 \to \beta_3,$$
$$\dots\dots\dots$$
$$\alpha_{2^k} \to \beta_{2^k},$$

де α_i, β_i – слова довжини, відповідно, k та n, $n > k$. Кажуть, що *схема* $\sigma_{k,n}$ *визначається кодом* $V = \{\beta_1, \beta_2, \beta_3, \dots, \beta_{2^k}\} \subset E_2^n$. Слова в алфавіті $\{0, 1\}$ – це впорядковані набори з нулів та одиниць (такі набори називають також двійковими векторами). Зазначимо, що кількість всіх різних слів α_i довжини k в алфавіті $\{0, 1\}$ дорівнює 2^k.

Виявлення однієї помилки

Спробуємо з'ясувати, чого можна сподіватись, коли до кожного елементарного коду додати всього лише один контрольний символ. Нехай кодування здійснюється за схемою

$$x_1x_2\ldots x_{n-1} \to x_1x_2\ldots x_{n-1}x_n.$$

Тоді контрольний символ x_n дорівнює нулю, якщо в повідомленні $x_1x_2\ldots x_{n-1}$ парна кількість одиниць, і одиниці – якщо непарна.

Формально це можна описати, використавши операцію $\oplus$ додавання за mod 2 для двійкових розрядів (бітів): $0\oplus0=0$, $0\oplus1=1$, $1\oplus0=1$, $1\oplus1=0$. Тоді елементарні коди містять парну кількість одиниць, тобто

$$x_1 \oplus x_2 \oplus \ldots \oplus x_{n-1} \oplus x_n = 0,$$

або, що те саме,

$$x_n = x_1 \oplus x_2 \oplus \ldots \oplus x_{n-1}.$$

Наприклад, приєднавши таким способом контрольний символ до слова 1010, одержимо слово 10100, а зі слова 1110 одержимо слово 11101. Неважко пересвідчитись, що всі елементарні коди містять парну кількість одиниць. Якщо під час передавання в елементарному коді відбудеться одна помилка (або навіть будь-яка непарна кількість помилок), то це буде виявлено, бо кількість одиниць у прийнятому елементарному коді буде непарною, але визначити, який саме розряд помилковий, неможливо. Потрібно повторити передавання елементарного коду, інакше помилки не виправити. Наприклад, якщо прийнято „неправильне слово" 11100, то однаково можливо, що було передано будь-який із елементарних кодів:

01100, 10100, 11000, 11110, 11101.

Кожний із зазначених випадків відповідає одній помилці, а насправді помилки могли відбутись навіть у трьох чи п'яти символах.

Ще гірше, коли відбудеться подвійна помилка чи взагалі парна кількість помилок. Тоді кількість одиниць у переданому елементарному коді залишиться парною, і помилковий елементарний код буде сприйнято як правильний.

Описаний вище код, який називають *кодом із загальною перевіркою на парність*, дає змогу визначити наявність непарної кількості помилок, але не виявляє помилки, якщо їх кількість парна.

Виправлення однієї помилки – один приклад коду Геммінга

Нехай повідомлення складаються з усіх можливих двійкових векторів довжиною чотири (кількість таких повідомлень дорівнює 16). Додавши п'ятий біт загальної перевірки на парність, можна виявити, але не виправити одну помилку в будь-якому розряді елементарного коду.

Знайдемо, яка мінімальна кількість контрольних символів необхідна для виправлення будь-яких помилок в одному розряді слова довжиною 4. Неважко переконатись, що двох додаткових символів для цього недостатньо. Спробуємо обійтись трьома перевірочними символами, тобто будемо використовувати для кодування повідомлень двійкові слова $x_1x_2x_3x_4x_5x_6x_7$ довжиною сім. Наше завдання – визначити, чи виникла помилка, і, якщо виникла, то в якому розряді. Але це те саме, що вказати одне з восьми чисел від 0 до 7 (0 відповідає відсутності помилки).

Отже, потрібно закодувати повідомлення з чотирьох бітів. Додамо до них ще три контрольних (або паритетних) біти. Це можна зробити по-різному. За контрольні вважатимемо біти в позиціях степенів двійки: 1, 2, 4. Отже, елементарний код виглядатиме так: $p_1p_2d_3p_4d_5d_6d_7$, де p_1, p_2, p_4 – контрольні біти, а d_3, d_5, d_6, d_7 – інформаційні, тобто біти даних. Контрольні біти тут відповідають за різні комбінації інформаційних бітів і визначаються так:

$$p_4 = d_5 \oplus d_6 \oplus d_7,\ p_2 = d_3 \oplus d_6 \oplus d_7,\ p_1 = d_3 \oplus d_5 \oplus d_7.$$

Метод побудови цих рівностей такий. Усі символи першої рівності в двійкових записах номерів розрядів мають 1 в першому біті: p_{100}, d_{101}, d_{110}, d_{111}, причому номер p_4 є степенем двійки. Усі символи другої рівності у двійкових записах номерів розрядів мають 1 в другому біті, а третьої рівності – у третьому біті, причому номери контрольних розрядів p_2 та p_1 – також степені двійки. Цей метод загальний, і його можна використати для всіх двійкових векторів довжиною $n = 2^m - 1$, де розряди 1, 2, 4, …, 2^{m-1} (степені двійки) – контрольні, а решта $2^m - 1 - m$ розрядів – інформаційні (тоді буде m перевірочних рівностей).

Якщо тепер потрібно з'ясувати, чи відбулася під час передавання слова $p_1 p_2 d_3 p_4 d_5 d_6 d_7$ помилка в одному з символів p_4, d_5, d_6, d_7, то для цього досить обчислити суму

$$s_1 = p_4 \oplus d_5 \oplus d_6 \oplus d_7.$$

Якщо $s_1 = 1$, то відповідь „так", якщо $s_1 = 0$ – „ні". У випадку „так" перевіримо, чи немає помилки в символах d_6, d_7, а у випадку „ні" – чи немає помилки в символах p_2, d_3. У кожному з цих випадків відповідь дає значення суми

$$s_2 = p_2 \oplus d_3 \oplus d_6 \oplus d_7.$$

Наприклад, якщо значення обох сум s_1 та s_2 дорівнюють 1, то помилка є або в d_6, або в d_7. Усього є чотири комбінації значень сум s_1, s_2; їх наведено в табл. 1.

Таблиця 1

s_1	s_2	Місце помилки
1	1	d_6 або d_7
1	0	p_4 або d_5
0	1	p_2 або d_3
0	0	немає помилки або p_1

Нарешті, в кожному з чотирьох випадків потрібно вибрати одну з двох можливостей. Зробити це дає змогу значення суми

$$s_3 = p_1 \oplus d_3 \oplus d_5 \oplus d_7 .$$

Отже, маємо три перевірочних співвідношення:

$$\begin{aligned} s_1 &= p_4 \oplus d_5 \oplus d_6 \oplus d_7 = 0, \\ s_2 &= p_2 \oplus d_3 \oplus d_6 \oplus d_7 = 0, \\ s_3 &= p_1 \oplus d_3 \oplus d_5 \oplus d_7 = 0, \end{aligned}$$

які дають змогу або визначити, що помилки немає, або однозначно зазначити її місце. Якщо відбулася одна помилка, то її положення визначає число з двійковим записом $s_1 s_2 s_3$. Нехай наприклад, $s_1 = 1$, $s_2 = 0$, $s_3 = 1$. За табл. 1 доходимо висновку, що помилка відбулася в четвертому чи п'ятому розрядах; оскільки $s_3 = 1$, то вона – у п'ятому розряді, але $s_1 s_2 s_3 = 101$ як раз і являє собою двійковий запис числа 5.