«Прикарпатський національний університет імені Василя Стефаника»

Факультет математики та інформатики

Кафедра алгебри та геометрії

ДИПЛОМНА РОБОТА

на здобуття першого (бакалаврського) рівня вищої освіти

на тему «Основи теорії кодування. Програмна реалізація деяких алгоритмів»

Виконала: студенка IV курсу

Групи М-41

Спеціальності Е-7 «Математика»

Гасюк Н.В

Керівник: к.ф.-м.н., асист. Микицей О. Я.

Рецензент: к.ф.-м.н., доц. Мазуренко Н. І

Івано-Франківськ 2025 р

АНОТАЦІЯ

до дипломної роботи на здобуття першого (бакалаврського) рівня вищої освіти

Гасюк Надії Василівни

на тему:

«Основи теорії кодування. Програмна реалізація деяких алгоритмів.»

Дипломна робота присвячена вивченню основ теорії кодування як однієї з ключових складових сучасних інформаційних технологій. Метою дослідження є аналіз математичних моделей, що лежать в основі процесів стискання та захисту інформації, а також практична реалізація ефективних алгоритмів кодування.

У роботі розглянуто базові поняття інформації та алфавітного кодування, умови однозначності декодування, принципи побудови оптимальних кодів, зокрема кодів Шеннона-Фано та Хаффмана. Окрему увагу приділено кодам, стійким до перешкод, зокрема коду Хемінга, що дозволяє виявляти й виправляти помилки при передачі даних. У практичній частині реалізовано програму для кодування за алгоритмом Хаффмана, проведено аналіз ефективності побудованих кодів на прикладах.

Результати дослідження можуть бути застосовані у сфері передавання, стискання та зберігання цифрової інформації, а також в освітньому процесі для формування базових компетентностей у галузі дискретної математики та інформатики.

Загальний обсяг роботи становить 46 сторінок, з яких основний текст – 42.

Ключові слова: теорія кодування, інформація, префіксні коди, код Хаффмана, код Хемінга, завадостійке кодування, оптимальність, реалізація алгоритмів.

ABSTRACT

to the bachelor's thesis for obtaining the first (bachelor's) level of higher education

by Nadiia Vasylivna Hasiuk

on the topic:

“Fundamentals of Coding Theory. Software Implementation of Some Algorithms”

The bachelor’s thesis is devoted to the study of the fundamentals of coding theory as one of the key components of modern information technologies. The aim of the research is to analyze mathematical models underlying data compression and protection processes, as well as to implement effective coding algorithms in practice.

The paper explores the basic concepts of information and alphabetic coding, conditions for unambiguous decoding, and principles for constructing optimal codes, particularly Shannon–Fano and Huffman codes. Special attention is given to error-correcting codes, especially the Hamming code, which allows for error detection and correction during data transmission. The practical part includes the development of a program implementing Huffman coding and an evaluation of code efficiency using real examples.

The results of the research can be applied in digital data transmission, compression, storage systems, and in the educational process for building core competencies in discrete mathematics and computer science.

The total length of the thesis is 46 pages, with 42 pages of main content.

Keywords: coding theory, information, prefix codes, Huffman code, Hamming code, error-correcting codes, optimality, algorithm implementation.

ЗМІСТ

ВСТУП.....5
1. ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ.....7
1.1 Інформація та кодування.....7
1.2 Алфавітне кодування.....8
1.3 Умови однозначності декодування.....10
2. ОПТИМАЛЬНЕ КОДУВАННЯ.....12
2.1 Код Шеннона-Фано.....13
2.2 Код Хаффмана.....14
2.3 Приклади застосування методів Шеннона-Фано та Хаффмана для кодування інформації.....16
3. КОДИ, СТІЙКІ ДО ПЕРЕШКОД.....20
3.1 Теоретичні основи завадостійкого кодування.....20
3.2 Код Хемінга.....27
4. ПРОГРАМНА РЕАЛІЗАЦІЯ.....36
ВИКОРИСТАНІ ДЖЕРЕЛА.....43
ДОДАТКИ.....45

ВСТУП

Передача та зберігання інформації – основа сучасних цифрових технологій. Щодня великі обсяги даних кодуються, стискаються, передаються та декодуються. Надійність цих процесів залежить від правильного вибору методів обробки. Через це кодування набуло великого значення в математиці, інформатиці та техніці.

Актуальність теми пояснюється зростанням потреб у стисканні даних без втрати інформації та захисті від помилок. Сучасні технології вимагають ефективних рішень для передачі навіть у складних умовах. Оптимальні й завадостійкі коди допомагають гарантувати якість сигналу. Тому дослідження в цій сфері є актуальним та практично корисним.

Об'єкт дослідження – математичні моделі представлення, стискання та захисту інформації.

Предмет дослідження – алгоритми побудови ефективних кодів, зокрема оптимальних і таких, що дозволяють виявляти й виправляти помилки.

Мета роботи – дослідити основи теорії кодування та її прикладні аспекти. Зокрема, проаналізувати принципи побудови алфавітних і завадостійких кодів, вивчити алгоритми Шеннона-Фано та Хаффмана, а також реалізувати один із них на практиці.

Завдання дослідження:

- вивчити базові поняття теорії кодування;
- розглянути приклади кодування з використанням префіксних структур;
- проаналізувати властивості коду Хемінга;
- реалізувати алгоритм Хаффмана засобами програмування;
- оцінити ефективність побудованих кодів на практичних прикладах.

Методи дослідження – теоретичний аналіз, моделювання, реалізація алгоритмів мовою програмування, а також математичні обчислення.

Практичне значення полягає у створенні програми, яка виконує кодування за алгоритмом Хаффмана. Це дозволяє оцінити роботу коду на реальних повідомленнях і зробити висновки про його ефективність.

Структура роботи. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У тексті містяться рисунки, приклади коду та результати обчислень.

1. ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Інформація та кодування

Кодування активно застосовується для вирішення завдань, які пов’язані з стисненням даних. Цей процес дозволяє зменшити обсяг інформації шляхом усунення надлишковості. Яскравим прикладом служить алгоритм DEFLATE, який здатний суттєво зменшити розмір файлів. Це важливо, особливо коли йдеться про обмежені ресурси, наприклад, пропускна здатність мережевого з’єднання. Проте, окрім стиснення, не менш важливим є і кодування, яке спрямоване на контроль помилок. Воно відіграє ключову роль у забезпеченні цілісності інформації при її передачі в умовах шуму або спотворень.

Перші теоретичні основи кодування пов’язують із роботами американського вченого Клода Шеннона. У 1948 році він опублікував фундаментальну наукову працю «Математична теорія зв’язку»[2]. Вона заклала основи теорії інформації як окремої наукової галузі. У своїй роботі Шеннон вперше показав, що проблему передачі повідомлень можна формалізувати за допомогою математичних моделей. Саме він увів такі базові поняття, як ентропія – міра невизначеності повідомлення, а також надлишковість, яка дозволяє підвищити надійність передачі, навіть коли канал зазнає перешкод. Ці ідеї стали фундаментом для створення більшості сучасних методів передавання даних у нестабільних чи зашумлених середовищах.

З точки зору формальної моделі, процес кодування зазвичай описується через поняття алфавітів. Припустимо, що маємо два алфавіти: $A = \{a_1, a_2, \ldots, a_n\}$ – початковий, та $B = \{b_1, b_2, \ldots, b_n\}$ – алфавіт, що використовується для кодування. Процес кодування визначається функцією $F: S \to B^*$, де S є підмножиною множини A^*. У цьому контексті A^* означає всі можливі слова, складені з елементів алфавіту A, а B^* – це множина всіх можливих послідовностей символів з алфавіту B, які утворюють коди. Таким чином, кожне

слово α з множини S за допомогою функції кодування перетворюється на унікальну послідовність β, яка і є його закодованою формою.

Завдяки такій моделі ми отримуємо можливість не лише описати процес кодування з математичної точки зору, а й використовувати його як основу для побудови ефективних алгоритмів – як для стиснення даних, так і для забезпечення їхньої достовірності при передаванні.

1.2 Алфавітне кодування

Алфавітне кодування є фундаментальним методом теорії інформації, який використовується для перетворення символів із заданого вихідного алфавіту у послідовності символів із цільового алфавіту. Основний принцип такого кодування полягає у побудові відповідного відображення між символами двох алфавітів, яке дозволяє кожному елементу вихідного алфавіту зіставити певне кодове слово з алфавіту-одержувача. Це дозволяє представити повідомлення у формі, зручній для подальшої обробки, передавання або зберігання.

Формально кодування описується математичним відображенням $F\colon A \to B^*$, де $A = \{a_1, a_2, \ldots, a_n\}$ – вихідний алфавіт, B – цільовий алфавіт, а B^* – множина всіх можливих скінченних послідовностей символів з B, включаючи порожнє слово. Кожному символу $a_i \in A$ ставиться у відповідність унікальне кодове слово $\beta_i \in B^*$, яке й використовується для кодування.

Такий підхід гарантує, що процес кодування є однозначним і ефективним, а декодування не викликає неоднозначності. Принцип однозначності декодування та умову префіксності детально розглянуто в підрозділі 1.3.

Першим кроком у побудові такого кодування є вибір вихідного алфавіту, що складається з символів, які необхідно закодувати, та вибір цільового алфавіту, символи якого використовуються для створення кодових слів. Наприклад, двійковий алфавіт $B = \{0, 1\}$ часто застосовується в цифрових пристроях через свою простоту та зручність у реалізації електронними сигналами.

Після цього виконується зіставлення між кожним символом вихідного алфавіту та відповідним кодовим словом з цільового. У випадку фіксованої довжини кодових слів усім символам призначаються слова однакової довжини. Щоб така схема працювала коректно, кількість можливих кодових слів має бути не меншою, ніж кількість символів у вихідному алфавіті. Це виражається умовою $m^l \geq n$, де m – потужність цільового алфавіту, n – кількість символів у вихідному алфавіті, а l – довжина кодового слова. Наприклад, якщо $A = \{a, b, c, d\}$, а $B = \{0, 1\}$, то можна використовувати довжину $l = 2$, оскільки $2^2 = 4$. У цьому випадку можливе таке зіставлення: $a \to 00,\ b \to 01,\ c \to 10,\ d \to 11$. Закодувавши повідомлення $M = abcd$, отримаємо кодову послідовність$E(M) = 00011011$. Фіксована довжина кодових слів полегшує процес декодування, оскільки дозволяє однозначно розпізнавати кожен символ. Проте такий підхід не враховує частотність появи символів, а отже, не завжди є оптимальним з погляду стискання інформації. Для підвищення ефективності часто використовують кодування зі змінною довжиною кодових слів. У таких схемах коротші коди призначаються частим символам, а довші – рідковживаним. Це дозволяє зменшити загальну довжину закодованого повідомлення, що особливо важливо при стисненні даних.

Окрім ефективного представлення символів, алфавітне кодування може виконувати ще одну важливу функцію – виявлення та виправлення помилок. Одним із найпростіших прикладів є біт парності, який дозволяє виявити одиничну помилку в кодовому слові. Цей біт показує, чи кількість одиниць у слові є парною чи непарною. У складніших схемах, таких як код Хеммінга, застосовуються додаткові контрольні біти, що не тільки сигналізують про помилку, а й дають змогу її виправити.

Таким чином, алфавітне кодування виконує ключову роль у цифрових технологіях, дозволяючи ефективно, надійно та однозначно представляти інформацію у вигляді послідовностей символів цільового алфавіту. Воно

використовується в найрізноманітніших сферах, від стискання даних до захисту від помилок, і є основою для створення складніших інформаційних систем.

1.3 Умови однозначності декодування

Однозначне декодування – це ключова умова для безпомилкового відновлення початкового повідомлення з його кодового представлення. Це спрощує процес декодування, оскільки дає змогу обробляти кодовану послідовність поетапно, без потреби у роздільниках або додатковому аналізі структури.

Необхідною і достатньою умовою для існування префіксного коду та однозначності кодування є нерівність Крафта-Макміллана[1]. Для множини кодових слів із довжинами $l_1, l_2, \ldots, l_n$, які будуються на основі алфавіту розміру r, нерівність виглядає так:

$$\sum_{i=1}^{n} r^{-l_i} \leq 1[1].$$

Ця нерівність показує, що сума ваги інформаційних блоків у кодовій системі не повинна перевищувати допустимого значення. У випадку, коли нерівність виконується зі строгим знаком можна додати кодові слова. Якщо сума дорівнює одиниці, код є повним. Це означає, що жоден новий елемент не може бути доданий без порушення умови однозначності.

Однозначність розшифрування кодового повідомлення забезпечується тим, що кожне кодове слово не може починатися з іншого, тобто система кодування є префіксною. Це означає, що декодування можна здійснювати послідовно зліва направо, без необхідності у розділових символах. Наприклад, при такому зіставленні символів: $a \to 0,\ b \to 10,\ c \to 110$ – послідовність 010110 легко розбивається на окремі кодові слова й однозначно перетворюється у «a, b, c». Саме така властивість дозволяє будувати ефективні схеми безпомилкового дешифрування.

Алгоритм процесу декодування полягає в тому, що якщо задано закодоване повідомлення $\beta = b_1, b_2 \ldots b_k$, що складається із символів кодового алфавіту,

завдання декодера полягає у розбитті цієї послідовності на кодові слова, кожне з яких відповідає одному символу вихідного алфавіту. У випадку префіксного кодування можна обробляти послідовність символ за символом зліва направо. Декодер знаходить найдовший префікс, який відповідає допустимому кодовому слову, декодує його, а потім переходить до решти послідовності. Формально, якщо знайдено префікс β', що належить образу функції кодування $F(A)$, то він може бути декодований, після чого аналіз переходить до залишку послідовності.

Процес декодування може потребувати додаткових перевірок та алгоритмів у складніших випадках для забезпечення правильного зчитування інформації. Наприклад, якщо схеми не є префіксними, то важливою є властивість самосинхронізації. Вона дозволяє декодеру автоматично відновлювати позицію у закодованій послідовності навіть після втрати синхронізації. Згідно з математичним твердженням, код вважається самосинхронізованим тоді, коли найбільший спільний дільник довжин усіх кодових слів дорівнює одиниці. Це означає, що після будь-якої помилки декодер зможе повернутися до правильного положення, як тільки зустріне валідне кодове слово.

У підсумку, саме поєднання префіксних кодів, критерію Крафта–Макміллана та здатності до самосинхронізації створює фундамент для надійного і простого декодування, що є важливою умовою для ефективного функціонування систем передачі та зберігання цифрової інформації.

2. ОПТИМАЛЬНЕ КОДУВАННЯ

У теорії інформації важливе місце займає задача побудови оптимальних кодів. Цей підхід широко використовується у сфері зв'язку, зберігання даних та алгоритмах стиснення інформації. Найчастіше такі коди реалізуються у вигляді префіксних.

Оптимальні схеми кодування повинні враховувати, що символи з вищою ймовірністю появи мають отримати коротші кодові слова, а ті, що трапляються рідше – довші. Інакше кажучи, якщо $p_i > p_j$, то повинно бути $l_i < l_j$. Недотримання цієї властивості призводить до збільшення середньої довжини повідомлення, що негативно позначається на ефективності системи.

Розглянемо алфавіт $A = \{a_1, a_2, ..., a_n\}$, що містить символи джерела, та відповідний розподіл ймовірностей $P = \{p_1, p_2, ..., p_n\}$, де кожне p_i відповідає символу a_i. При кодуванні кожному символу a_i зіставляється кодове слово, що складається з символів цільового алфавіту B. Довжина кожного з цих кодів позначається як l_i. Задача оптимального кодування полягає у мінімізації середнього значення довжин кодів, що математично подається формулою:

$$L = \sum_{i=1}^{n} p_i l_i \rightarrow min[5].$$

Цей вираз є критерієм, за яким оцінюється оптимальність побудованого коду. Він показує, скільки в середньому бітів потрібно для передачі одного символу. В ідеалі, середня довжина коду L має бути якомога ближчою до ентропії джерела H, яка відображає середню кількість інформації, що міститься у кожному символі. Формула для ентропії має вигляд:

$$H = -\sum_{i=1}^{n} p_i log_2 p_i[3].$$

Різниця між цими двома величинами – середньою довжиною L та ентропією H – називається надлишковістю, яка обчислюється за формулою:

$$R = L - H[4].$$

Чим менше значення R, тим ефективніше використовується кодовий алфавіт. У практиці трапляються ситуації, коли точно дотриматись умов неможливо – наприклад, через обмежену кількість символів у цільовому алфавіті або коли точні ймовірності невідомі. У таких випадках значення R не дорівнює нулю, але завданням є мінімізувати його настільки, наскільки це можливо.

Таким чином, оптимальні коди дозволяють досягти максимальної ефективності при зберіганні та передаванні інформації. Вони дають змогу зменшити обсяг переданих даних, знизити навантаження на систему та підвищити швидкість обробки. З огляду на це, оптимальне кодування відіграє визначальну роль у створенні ефективних систем зберігання, обробки та передавання інформації.

2.1 Код Шеннона-Фано

Алгоритм Шеннона-Фано, запропонований Клодом Шенноном і Робертом Фано, є прикладом практичного втілення ідей ефективного кодування змінної довжини[6].

Однією з ключових характеристик методу Шеннона-Фано є те, що він генерує префіксні коди, що гарантує однозначне декодування будь-якого повідомлення. У цьому методі частота появи символів прямо впливає на довжину кодів: ймовірніші символи отримують коротші представлення. Це сприяє зменшенню обсягу переданої або збереженої інформації.

Розглянемо основні кроки роботи цього методу. На першому етапі здійснюється сортування елементів алфавіту $A = \{a_1, a_2, ..., a_n\}$ у порядку спадання ймовірностей їх появи, заданих через множину $A = \{a_1, a_2, ..., a_n\}$, де p_i – ймовірність символу a_i. Це впорядкування є критичним, оскільки воно впливає на логіку подальшого присвоєння кодів і забезпечує основу для ефективної побудови префіксної структури.

Далі відбувається рекурсивне розбиття множини символів на дві частини – S_1 та S_2, – при цьому намагаються зробити так, щоб сума ймовірностей кожної з підмножин була максимально близькою до 50% від загальної. Це розбиття повторюється доти, поки кожна підмножина не міститиме лише один елемент.

На кожному кроці такого поділу кожній підмножині призначається бінарний символ: першій – «0», другій – «1». Таким чином, код для конкретного символу формується шляхом послідовного додавання (конкатенації) бітів, призначених на кожному рівні поділу. Формально це можна виразити наступним чином: якщо S – початкова множина, а S_1 і S_2 – результат поділу, то:

$$(S_1) = (S)0, \quad (S_2) = (S)1,$$

де (S) – код батьківської множини, а «0» та «1» – відповідні бітові значення для новоутворених підмножин.

Таким чином, кодування методом Шеннона-Фано дозволяє отримати унікальний бінарний код для кожного символу, враховуючи його частотність у джерелі. Створені коди завжди префіксні, що забезпечує їхню зручність для декодування без потреби в розділювачах або буферах.

Попри свою простоту та логічність, метод Шеннона-Фано не завжди гарантує абсолютно найкращу (мінімальну) середню довжину повідомлення, як це робить, наприклад, алгоритм Хаффмана. Попри це, завдяки наочності, чіткій структурі та простоті реалізації, метод Шеннона–Фано зберігає цінність як інструмент для навчання та аналізу ефективності методів стиснення даних.

2.2 Код Хаффмана

Алгоритм Хаффмана є одним із найпоширеніших рішень для побудови ефективного стиснення даних без втрати інформації. Розроблений Девідом Хаффманом у 1952 році, цей метод набув статусу класичного в теорії кодування[12]. На відміну від методу Шеннона-Фано, код Хаффмана будується за принципом мінімізації середньої довжини з урахуванням частотності, що забезпечує його перевагу в реальних застосуваннях.

У коді Хаффмана символи з більшою частотою появи представлені коротшими бінарними послідовностями, тоді як рідкісні символи кодуються довшими кодовими словами. Побудова коду виконується у два основні етапи. Перший – це створення бінарного дерева кодування, яке дістало назву дерево Хаффмана.

На початку роботи алгоритму усі символи та відповідні їм ймовірності формуються у список. Кожен символ виступає як окремий вузол дерева, при цьому вага вузла відповідає ймовірності появи символу. Подальші дії полягають у послідовному об'єднанні двох вузлів із найменшими вагами в новий вузол. Вага нового вузла дорівнює сумі ваг двох попередніх. Така операція виконується до того моменту, поки не залишиться лише один кореневий вузол, який об’єднує всю структуру.

Після завершення побудови дерева відбувається другий етап – призначення кодів. Для цього здійснюється обхід дерева від кореня до кожного листка. На кожному розгалуженні визначаються бітові значення «0» – якщо рухаємось ліворуч, «1» – якщо праворуч. У результаті кожен символ отримує унікальний бінарний код. Ті символи, які мають вищу ймовірність, розташовуються ближче до кореня дерева і, відповідно, мають коротші коди, а символи з меншою ймовірністю – глибше в дереві та довші послідовності бітів.

Ефективність алгоритму Хаффмана можна оцінити за допомогою середньої довжини кодових слів L, яка визначається так само, як і у методі Шеннона-Фано. При цьому значення L наближається до ентропії джерела H, яка є нижньою теоретичною межею:

$$L \approx H$$

Завдяки цій властивості, алгоритм Хаффмана дозволяє мінімізувати надлишковість, що робить його надзвичайно привабливим для застосування у реальних системах.

Метод Хаффмана використовується у великій кількості технологій стиснення, серед яких найвідоміші – формати JPEG, MPEG, ZIP та інші. Простота реалізації та висока продуктивність забезпечили йому довготривале

лідерство серед алгоритмів кодування. Водночас, найкращі результати досягаються тоді, коли ймовірності символів відомі наперед і залишаються постійними. У ситуаціях, коли частотні характеристики змінюються динамічно, використовуються адаптивні версії алгоритму Хаффмана, які можуть підлаштовуватись до нового розподілу ймовірностей у процесі кодування.

2.3 Приклади застосування методів Шеннона-Фано та Хаффмана для кодування інформації

Для демонстрації практичного застосування алгоритмів Шеннона-Фано та Хаффмана візьмемо слово «$MATHEMATICS$», що складається з 11 символів. На першому етапі необхідно підрахувати частоту появи кожної літери в слові:

M – 2

A – 2

T – 2

H – 1

E – 1

I – 1

C – 1

S – 1

Після цього визначаємо ймовірність появи кожного символу у повідомленні:

$$P(M) = \frac{2}{11} \approx 0,18$$

$$P(A) = \frac{2}{11} \approx 0,18$$

$$P(T) = \frac{2}{11} \approx 0,18$$

$$P(H) = \frac{1}{11} \approx 0,09$$

$$P(E) = \frac{1}{11} \approx 0,09$$

$$P(I) = \frac{1}{11} \approx 0,09$$

$$P(C) = \frac{1}{11} \approx 0,09$$

$$P(S) = \frac{1}{11} \approx 0,09$$

Виконаємо кодування методом Шеннона-Фано:

На основі отриманих ймовірностей виконується побудова коду за алгоритмом Шеннона-Фано. Спершу символи впорядковуються за спаданням ймовірностей. Потім множина ділиться на дві підгрупи з приблизно рівними сумарними ймовірностями. Кожній з груп присвоюється біт – 0 або 1:

група 1: $T,\ M,\ A$ (сумарна частота = 6);

група 2: $S,\ I,\ H,\ E,\ C$ (сумарна частота = 5).

Далі кожну підгрупу ділять аналогічним чином, поки не залишиться лише один символ. Кодові слова формуються шляхом конкатенації бітів, які було призначено на кожному кроці поділу:

$$T,\ M,\ A \rightarrow 0;$$

$$S,\ I,\ H,\ E,\ C \rightarrow 1;$$

$$A \rightarrow 00;\ M,\ T \rightarrow 01;$$

$$M \rightarrow 010;\ T \rightarrow 011;$$

$$S \rightarrow 1111;\ I,\ H,\ E,\ C \rightarrow 1110;$$

$$H \rightarrow 110;\ E \rightarrow 101;\ I \rightarrow 1110;\ C \rightarrow 100.$$

Після завершення процедури отримаємо набір префіксних кодів, де найбільш ймовірні символи мають коротші коди. Така структура забезпечує ефективне та однозначне декодування.

01000011110101010000111110100111 1.

Перейдемо до кодування методом Хаффмана:

Алгоритм Хаффмана, який детально описаний у підрозділі 2.2, використовується для створення ефективного кодування з урахуванням того, як часто зустрічаються символи у повідомленні. У цьому прикладі ми беремо ймовірності появи кожної букви та на їх основі будуємо дерево Хаффмана.

Під час побудови дерева найрідші символи розміщуються глибше, а найчастіші – ближче до кореня. Завдяки цьому найуживаніші символи

отримують найкоротші коди. Це дозволяє значно скоротити загальну довжину закодованого тексту.

Отриманий набір кодів є префіксним, тобто жодне кодове слово не є початком іншого, і це забезпечує однозначне декодування.

Приклад сформованого коду та результат кодування слова наведено на рисунку 1.

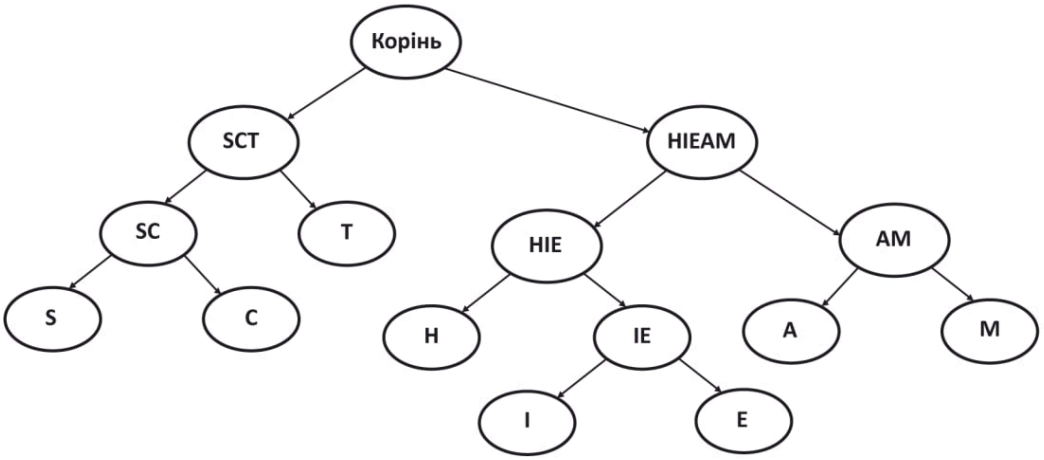

Рисунок 1

У підсумку обидва методи забезпечують створення ефективного кодування для заданого слова, проте алгоритм Хаффмана в більшості випадків дозволяє досягти меншої середньої довжини коду. У прикладі з «MATHEMATICS» це також підтверджується результатами побудованих кодів.

Закодоване слово методом Хаффмана:

11111001100101111111100110100010000.

Виконаємо декодування:

Щоб розшифрувати отримане повідомлення, потрібно зчитувати біти один за одним і зіставляти їх із закодованими комбінаціями, які відповідають кожному символу. Коли послідовність бітів збігається з одним із кодів,

розшифрований символ додається до результату. Розглянемо приклад декодування коду Хаффмана:

$$111 \to M,$$
$$110 \to A,$$
$$01 \to T,$$
$$100 \to H,$$
$$1011 \to E,$$
$$111 \to M,$$
$$01 \to T,$$
$$110 \to A,$$
$$1010 \to I,$$
$$001 \to C,$$
$$000 \to S.$$

Результат: $MATHEMATICS$.

3. КОДИ, СТІЙКІ ДО ПЕРЕШКОД

3.1 Теоретичні основи завадостійкого кодування

Навіть за умов розвитку цифрових технологій, ймовірність виникнення помилок при передачі даних залишається постійним викликом для інженерії зв’язку. Через технічні чи зовнішні перешкоди сигнал у каналі зв’язку може зазнавати спотворень або навіть повністю втрачатися.

У деяких умовах, як-от під час міжпланетних місій, повторне звернення до джерела неможливе, тому система має працювати автономно. Для подолання таких проблем розроблено коди, здатні самостійно ідентифікувати та коригувати спотворення в даних. За рахунок додавання контрольної надлишкової інформації сучасні методи кодування значно підвищують достовірність переданих даних.

Навіть одиничні помилки, які не помітні на рівні сигналу, можуть призвести до втрати цілісності або зміни сенсу переданої інформації. У таких випадках використовують завадостійке кодування, щоб мати можливість коректно передавати дані.

У деяких системах коди з корекцією помилок вбудовуються безпосередньо в мікросхеми, що відповідають за зчитування та обробку даних. Також їх можуть інтегрувати в програмне забезпечення, де імітується процес перевірки цілісності.

Таким чином, вся сучасна концепція захисту інформації в цифровому середовищі значною мірою базується на розробці ефективних способів боротьби з перешкодами. Неможливість повністю уникнути помилок зумовлює потребу створювати методи виявлення і корекції, які можуть не лише забезпечувати точність, але й дозволяють суттєво зменшити обсяг надлишкових даних. Саме тому ці методи стали базовими інструментами для забезпечення надійності цифрових систем.

Щоб ефективно протистояти викривленню даних, потрібно чітко розуміти природу і класифікацію можливих помилок. Перш за все, варто звернути увагу

на випадкові помилки, які ще також називають односимвольними. Зазвичай вони виникають через короткочасний вплив перешкод у каналі зв’язку. Вони проявляються у вигляді одиничного спотворення символу або біта. Прикладом може бути зміна одного біта з 0 на 1 або навпаки. Такі помилки часто поширені в практиці, особливо при роботі з нестабільними каналами, де канал часто слабшає.

Також поширеним типом є групові або пакетні помилки. Вони виникають внаслідок сильніших або довших за тривалістю завад і охоплюють одразу кілька сусідніх символів. На відміну від одиничних, ці помилки важче виявити і виправити, особливо якщо спотворення зачіпає логічну структуру кодованих даних.

Бувають також симетричні та асиметричні помилки. Симетричні означають, що ймовірність перетворення «1» в «0» і «0» в «1» однакова, а асиметричні мають нерівномірний характер. Наприклад, частіше трапляються помилку типу 0 в 1, ніж 1 в 0 у певних фізичних умовах. Тип помилки залежить від властивостей передавального середовища, технічних параметрів обладнання щодо. Асиметричні помилки складніші для обробки, оскільки вимагають спеціально адаптованих кодових структур.

Перешкоди також можуть бути сталими, короткочасними, імпульсними та періодичними. Сталі – це помилки, що повторюються в тому самому місці або виникають на постійно несправній ділянці пристою. Короткочасні – це одиничні порушення, які не мають чіткої закономірності. Імпульсні помилки спричиняються короткочасним впливом, а періодичні ж виникають із певною циклічністю, яка часто пов’язана із коливанням живлення або роботою двигунів.

Розуміння характеру і типу помилки є фундаментальним етапом при побудові будь-якої схеми кодування, стійкого до перешкод. Лише маючи чітке уявлення про природу потенційних викривлень, можна створити систему, здатну гарантувати достовірність переданої інформації.

У передаванні цифрової інформації такоє є надзвичайно важливим вміння моделювати сам канал зв’язку, в якому виникають ці помилки. Необхідно мати математичну модель для побудови ефективних механізмів виявлення та виправлення помилок, яка буде відображати поведінку каналу. Така модель дозволяє оцінити ризики втрати даних і визначити межу до якої канал здатний передавати інформацію з прийнятним рівнем надійності.

У фундаменті будь-якої системи захисту інформації лежить поняття надлишковості. Це одна з ключових ідей у завадостійкому кодуванні. Вона полягає у свідомому додаванні до повідомлення певної кількості додаткових символів, які не несуть нового змісту, але мають чітко визначену математичну структуру. Ці символи генеруються за строгими правилами на основі інформаційної частини повідомлення і дозволяються перевірити наявність спотворення в ході передавання. Надлишковість відкриває можливість побудувати простір кодових слів з певною відстанню між ними. Саме ця відстань визначає здатність системи активно боротись з помилками.

Відстань між кодовими словами зазвичай є Хемінговою[9]. Вона дозволяє формально задати умови виявлення та виправлення помилок. Якщо мінімальна відстань між двома допустимими кодовими словами дорівнює d, то система може виявити до $d - 1$ помилок, а також виправити до $t = \lfloor \frac{d-1}{2} \rfloor$ помилок. При отриманні закодованого повідомлення приймач порівнює його з усіма можливими правильними кодовими словами і визначає найближче з метрикою Хемінга. Якщо відстань між ними достатньо велика, можна бути впевненим, що навіть у разі кількох помилок повідомлення не буде прийнято за інше. Тобто не станеться некоректного декодування.

Проте, через цей механізм може бути зниження інформаційної ефективності. Якщо позначити довжину кодового слова як n, а кількість інформаційних бітів як k, то відношення $R = \frac{k}{n}$ називається швидкістю коду[7]. Чим більше надлишкових бітів додається, тим потужнішим стає захист.

Але в той же час зменшується кількість корисної інформації, яку можна передати за одиницю часу.

Саме тому ключовим завданням теорії кодування є пошук такої структури коду, яка при мінімальному обсязі надлишкової інформації буде забезпечувати заданий рівень захисту. Застосовують два принципово різні підходи в залежності від задачі. Іноді достатньо лише виявляти помилки і не потрібно вміти їх виправляти. Наприклад, у тих системах, де можливе повторне передавання повідомлення, наприклад у протоколах TCP/IP. Як тільки приймач виявляє помилку – він запитує повтор. Для цього існують прості й ефективні методи, такі як парність або CRC.

В умовах, коли повторити передачу неможливо і потрібно мати змогу відновити дані автономно, застосовуються вже інші коди. Вони складніші за структурою та дозволяють не лише констатувати факт помилки, а й точно вказати, які біти слід змінити. Такі коди потребують глибшого математичного апарату, часто пов’язаного з алгеброю над полями, побудовою матриць, розв’язуванням систем рівнянь, що описують структуру помилки.

Основна ціль таких кодів – гарантувати збереження даних попри спотворення в каналі зв’язку. Критично важливими відіграють роль такі елементи, як довжина слова, кількість контрольних бітів, спосіб їх розрахунку.

Важливим є також класифікація кодів, стійких до перешкод. Їх можна впорядкувати за кількома базовими критеріями, які дозволяють швидко визначити, який тип коду найкраще підходить для конкретної практичної задачі.

Один із найочевидніших критеріїв класифікації базується на структурі побудови кодових слів. Виділяють блокові та згорткові коди. Блокові коди – це відображення $f : F_q^k \to F_q^n$, де F_q – скінченне поле порядку q, k – кількість інформаційних символів, а n – загальна довжина кодового слова. Такі коди працюють із фіксованими блоками інформації, кожен з яких незалежно перетворюється у кодову комбінацію. Це дозволяє застосовувати до кожного

блоку однакові правила кодування та декодування, а також використовувати матричні методи для аналітичного опису коду.

Також виділяють згорткові коди. Вони оперують не окремими блоками, а потоком символів. Кожен вихідний символ є функцією кількох попередніх, а не лише поточного вхідного символу. Такий код задається як лінійний перетворювач над простором формальних степеневих рядів або за допомогою скінченного автомату. Особливістю таких кодів є наявність пам'яті, що ускладнює декодування, але забезпечує високу ефективність при обробці безперервного потоку даних.

Важливим аспектом класифікації також є лінійність. Якщо множина кодових слів утворює підпростір у F_q^n, тобто замкнена відносно додавання та множення на скаляри з поля F, то такий код називається лінійним. Нелінійні коди навпаки можуть мати більшу мінімальну відстань при однаковій довжині та ефективності.

Лінійні блокові коди посідають центральне місце в теорії завадостійкого кодування. Будь-який лінійний блоковий код C над скінченним полем F_q визначається як k-вимірний підпростір простору F_q^n, де n – довжина кодового слова, а k – кількість інформаційних символів. Це означає, що кожне кодове слово $c \in C$ є вектором у F_q^n, який породжується лінійною комбінацією k базисних векторів. Відповідно, весь код Cскладається з q^k таких векторів. Саме таке представлення дозволяє аналізувати коди як геометричні об'єкти у векторному просторі, де структура визначається параметрами (n, k, d), і зокрема – відстанню d, яка гарантує їхню здатність до виявлення та виправлення помилок.

Кодування у лінійному коді виконується через множення вектора повідомлення $u \in F_q^k$ на матрицю генератора $G \in F_q^{k \times n}$. В результаті отримується кодове слово $c = uG$, яке належить простору C. Матриця G є

фундаментальним об'єктом, оскільки її рядки утворюють базис підпростору C. У випадку систематичної форми, яка особливо зручна на практиці, матриця G має вигляд $G = \left[I_k \mid P\right]$, де I_k – одинична матриця, а P – частина, яка відповідає за контрольні символи. Це дозволяє безпосередньо включати інформаційні біти до структури кодового слова, що спрощує як кодування, так і декодування.

Щоб реалізувати перевірку правильності прийнятого повідомлення, вводиться перевіркова матриця $H \in F_q^{(n-k)\times n}$, яка є ортогональною до простору C. Тобто для будь-якого кодового слова $c \in C$ виконується рівність $Hc^T = 0$. Всі вектори, які задовольняють цю умову, утворюють ядро матриці H, яке й є самим кодом. Іншими словами, $C = ker\, H$, і це дає альтернативне, але еквівалентне означення лінійного коду як ядра лінійного відображення. Побудова H зазвичай виконується так, щоб виконувалась умова $GH^T = 0$, що гарантує ортогональність простору кодових слів і простору перевірки.

У випадку, коли повідомлення проходить через зашумлений канал, на приймач надходить вектор $r = c + e$, де e – вектор помилок, що має ненульові координати в позиціях спотворених символів. Для виявлення помилки обчислюється синдром $s = Hr^T$. Якщо $s = 0$, це свідчить про відсутність помилки або про те, що вектор помилок лежить у кодовому просторі, що трапляється лише в крайніх випадках. Якщо ж $s \neq 0$, це однозначно свідчить про наявність спотворення. Сам синдром несе в собі інформацію про помилку, і його аналіз дозволяє локалізувати та виправити помилки при відомому алгоритмі декодування.

Уся структура лінійного коду базується на концепції векторного підпростору. Кодове слово розглядається не як абстрактна послідовність, а як елемент певного геометричного простору, у якому працюють лінійні операції, і для якого можна визначити базис, ортогональний комплементарний простір, ядро та образ відображення. Такий підхід дозволяє гнучко працювати з

просторами великої розмірності, розробляти алгоритми ефективного декодування, використовувати методи оптимізації, пов’язані з розміщенням векторів у просторі, та контролювати обчислювальну складність операцій.

Таким чином, основні характеристики лінійних блокових кодів базуються на трьох ключових математичних поняттях. Перш за все, це простір кодових слів. Його будова визначається генераторною матрицею, яка використовується для формування допустимих кодів. Другим важливим елементом є перевіркова матриця. Вона дозволяє перевірити, чи належить отримане слово до заданого коду. І нарешті, третій компонент – це синдром. Саме він дає змогу виявляти помилки та визначати їхню позицію. Завдяки такій чіткій структурі вдається ефективно реалізовувати алгоритми виявлення та виправлення помилок у реальних інформаційних системах.

Завершуючи теоретичний розгляд завадостійких кодів, доцільно узагальнити основні висновки та чітко визначити роль цієї концепції в сучасній інформаційній парадигмі. Як показано в попередніх розділах, надійне передавання даних у присутності шуму, завад або втрат є не винятковим випадком, а типовим режимом функціонування реальних каналів зв’язку. Саме тому завадостійке кодування стало не просто частиною теорії інформації, а фундаментальною передумовою побудови всіх без винятку цифрових систем комунікації та зберігання. Його основне завдання полягає в тому, щоб компенсувати обмеження фізичного середовища за допомогою математично вивірених структур, які дозволяють гарантувати достовірність інформації навіть тоді, коли канал поведеться непередбачувано.

З математичної точки зору, завадостійке кодування поєднує у собі дисципліни, що охоплюють алгебру, комбінаторику, теорію ймовірностей та оптимізацію. У цій моделі дані розглядаються не як абстрактна послідовність символів, а як елементи алгебраїчних структур, над якими здійснюються чітко визначені операції. Генераторні матриці, перевіркові матриці, синдроми, відстані Хемінга – всі ці поняття надають системі гнучкості, керованості та здатності до самокорекції. І саме ця здатність до самостійного виправлення або

виявлення помилок без потреби у зворотному каналі є тим чинником, який забезпечує принципово новий рівень автономності в сучасних технологіях.

Практичне значення таких кодів було продемонстровано у цілої низці прикладів: від міжпланетного зв'язку до щоденного сканування QR-коду на етикетці. Без завадостійкого кодування було б неможливо зберігати дані у пам'яті комп'ютерів, передавати відео з високою роздільністю через мобільну мережу або записувати інформацію на компакт-диски. Той факт, що користувачі не помічають втрат або викривлень, є прямим свідченням того, що система кодування виконує свою функцію правильно і непомітно. Надійність у такому випадку – це не відсутність помилок, а їхня грамотна обробка.

Таким чином, розділ, присвячений загальній теорії кодів, стійких до перешкод, сформував базу для розуміння ключових понять і принципів побудови таких кодів. У ньому було розкрито як фундаментальні теоретичні аспекти, так і їхню практичну реалізацію. Наступним логічним кроком є перехід до вивчення конкретних прикладів, що ілюструють застосування розглянутих принципів на практиці. Одним із перших практичних прикладів реалізації завадостійкого кодування є код Хемінга, який чітко демонструє принципи виявлення та виправлення помилок. Його аналіз дозволить з практичної точки зору побачити, як працює завадостійке кодування на рівні окремих бітів, які обчислення лежать в основі виявлення та виправлення помилок, і які обмеження накладає структура коду на його можливості. Це відкриває новий етап дослідження – від загальної теорії до прикладної реалізації.

3.2 Код Хемінга

У сучасній теорії кодування одне з провідних місць займає код Хемінга – клас лінійних бінарних кодів, який дозволяє ефективно виявляти та виправляти одиночні помилки. Вперше його запропонував Річард Хемінг у 1950 році[9]. Цей код ліг в основу багатьох методів контролю помилок. Він залишається затребуваним завдяки своїй простій математичній структурі та ефективності в умовах завад. На відміну від загальних блокових кодів, Хемінгів код

побудований так, щоб за мінімальної надмірності забезпечити здатність до виявлення і точного локалізованого виправлення помилки у кодовому слові.

Код Хемінга належить до систематичних лінійних кодів, де інформаційні біти розміщуються на чітко визначених позиціях, а контрольні біти займають позиції з номерами, що є ступенями двійки (1, 2, 4, 8,…). Завдяки цій структурі досягається можливість однозначного визначення позиції помилки у разі її виникнення. Цю властивість забезпечує так званий синдром – двійковий вектор, який обчислюється на приймальному боці і відповідає номеру помилкового біта. Ця методика дозволяє не лише виявити, що передача порушена, але й однозначно вказати, де саме виникла помилка.

З математичної точки зору, код Хемінга будується за умови виконання співвідношення $2^r \geq n + 1$, де r – кількість перевірочних бітів, а n – загальна довжина кодового слова, що включає і інформаційні, і контрольні біти. Це забезпечує унікальність двійкових кодів для кожної можливої помилки, включаючи варіант її відсутності. Найпоширенішим представником є код (7, 4), де з 7 бітів чотири є інформаційними, а три – контрольними[8].

Окрім можливості виправляти одиничні помилки, код Хемінга демонструє і високу ефективність при виявленні подвійних помилок, хоча останні він виправити вже не здатен. З цією метою була запропонована модифікація – розширений код Хемінга з мінімальною кодовою відстанню $d_{min} = 4$, який додає ще один контрольний біт для детектування парної кількості помилок. Незважаючи на більшу надмірність, така модифікація є надзвичайно цінною в системах, що потребують високої надійності передачі інформації, зокрема в пристроях пам’яті типу ECC або в масивах RAID 2.

Таким чином, код Хемінга являє собою вдале поєднання математичної простоти, логічної прозорості та практичної ефективності. Його універсальність дозволила застосовувати цей код не лише в академічних дослідженнях, а й у багатьох галузях інформаційних технологій, де існує потреба у захисті даних від одиничних похибок. У наступних розділах буде детально розглянуто

механізм побудови та функціонування цього коду, а також приклади його реалізації на практиці.

Для реалізації коду Хемінга на практиці необхідно розуміти конкретні кроки побудови його кодових слів. У фокусі уваги цього пункту – процес формування (n, k) - коду, який передбачає чітке розміщення інформаційних та перевірочних бітів, побудову відповідних матриць та обчислення контрольних сум. Основна мета – забезпечити таку структуру коду, яка дозволяє ефективно виявляти і виправляти одну помилку у будь-якому розряді.

Початковим етапом побудови є визначення кількості перевірочних бітів. Це досягається шляхом розв'язання нерівності:

$$2^r \geq n + 1[8],$$

де $n = k + r$ – повна довжина кодового слова. Наприклад, для повідомлення з чотирма інформаційними бітами (тобто $k = 4$) отримуємо $r = 3$, а отже, довжина кодового слова становить $n = 7$. Відповідно, для $(7, 4)$ -коду позиції 1, 2 і 4 призначаються для контрольних бітів, а всі інші – для інформаційних. Варто наголосити, що індекси перевірочних позицій завжди дорівнюють ступеням двійки.

Після визначення структури кодового слова виконується його заповнення. Інформація заповнює решту позицій, не зарезервованих для перевірочних бітів. Контрольні біти на цьому етапі залишаються порожніми, оскільки вони будуть обчислені далі. Для цього застосовується система перевірочних рівнянь, що базується на побудові перевірочної матриці H розмірності $r \times n$, у якій кожен стовпець є двійковим представленням номера відповідного біта[10]. Ключовою вимогою є те, що всі стовпці цієї матриці мають бути лінійно незалежними та ненульовими.

Розв'язанням системи перевірочних рівнянь визначаються значення контрольних бітів – вони обчислюються як сума за модулем 2 відповідних інформаційних бітів. Наприклад, у класичному $(7, 4)$ - коді перший контрольний біт покриває біти 1, 3, 5 і 7; другий – 2, 3, 6 і 7; третій – 4, 5, 6 і 7. Таким чином, кожна контрольна сума забезпечує контроль над певною

множиною позицій, що дозволяє локалізувати помилку за допомогою обчислення синдрому при прийомі повідомлення.

Зі структурної точки зору побудова коду Хемінга також пов’язана з використанням т. зв. твірної матриці G, яка дозволяє безпосередньо здійснювати перетворення інформаційного вектора у закодоване повідомлення. Вона має розмірність $k \times n$ і формується таким чином, щоб кожен інформаційний біт копіювався в потрібну позицію, а контрольні біти обчислювались згідно з раніше сформованими перевірочними рівняннями. Наприклад, для $(7, 4)$ - коду твірна матриця має вигляд, при якому перші стовпці формують одиничну підматрицю, а решта – містить значення, пов’язані з перевірочними співвідношеннями.

Таким чином, побудова кодових слів у коді Хемінга ґрунтується на суворій логіці розміщення бітів і використанні чітких алгебраїчних структур. Завдяки цьому вже на етапі формування коду закладається можливість подальшого виявлення і виправлення помилок при передачі інформації. У наступних частинах буде проаналізовано, як відбувається саме декодування та пошук помилок за допомогою синдрому.

Процес кодування в коді Хемінга полягає у перетворенні вхідного інформаційного повідомлення у закодовану послідовність, яка містить додаткові перевірочні біти, що дозволяють забезпечити контроль цілісності даних. На цьому етапі вже відомі всі параметри коду – довжина кодового слова, кількість контрольних бітів, розподіл позицій. Ключовим завданням є правильне заповнення контрольних розрядів таким чином, щоб сформоване кодове слово задовольняло систему перевірочних рівнянь, тобто належало до підпростору допустимих кодових комбінацій.

Перед початком обчислень інформаційні біти послідовно записуються у відповідні позиції кодового слова, які не є ступенями двійки. Після цього, для кожного контрольного біта визначається набір позицій, що підлягає перевірці. Цей вибір ґрунтується на двійковому представленні номерів позицій: контрольний біт відповідає за ті розряди, у яких відповідний біт у двійковому

номері є одиницею. Наприклад, якщо розглядається контрольний біт, що стоїть у позиції 2, тобто 2 дорівнює 010 у двійковій системі, то він охоплює всі розряди, номери яких мають одиницю у другому зліва біті свого двійкового представлення.

Контрольні біти обчислюються як сума за модулем 2 усіх інформаційних бітів, що входять у їхню зону контролю. Це означає, що кожен контрольний біт – це результат операції XOR над відповідними значеннями. Завдяки такому підходу забезпечується парність у всіх контрольованих групах, і в разі, якщо вхідне повідомлення було передано без помилок, усі перевірочні рівняння виконуватимуться.

Щоб конкретизувати цей процес, розглянемо приклад кодування для класичного $(7, 4)$ - коду Хемінга. Нехай вхідне повідомлення має вигляд 1, 0, 1, 1. Згідно з розміткою позицій, ці чотири біти займають позиції 3, 5, 6 і 7 відповідно. У відповідних місцях розміщуються інформаційні біти, тоді як контрольні розряди в позиціях 1, 2 і 4 залишаються тимчасово незаповненими. Далі для кожного з них виконується підрахунок згідно з правилами перевірки: біт у позиції 1 охоплює позиції 1, 3, 5, 7; біт у позиції 2 – позиції 2, 3, 6, 7; біт у позиції 4 – позиції 4, 5, 6, 7. Підставивши відомі значення інформаційних бітів у ці рівняння, легко знайти значення кожного контрольного біта, які забезпечують виконання паритету в усіх групах.

У результаті отримаємо повне кодове слово, яке вже можна передавати по каналу. Його структура зберігає всі інформаційні біти у незмінному вигляді, що робить декодування інтуїтивно зрозумілим і зручним для реалізації. Таким чином, процес кодування в коді Хемінга зводиться до чітко визначених кроків: заповнення інформаційних позицій, побудови логічної структури зв’язків між бітами і обчислення контрольних розрядів відповідно до алгебраїчних співвідношень. Результатом є сформоване кодове слово, яке володіє необхідними властивостями для подальшого виявлення і виправлення помилок. У наступному розділі буде розглянуто, яким чином ця структура дозволяє проводити декодування та корекцію похибок.

Після того як кодове слово сформовано та передано по каналу, ключовим завданням на приймальному боці стає виявлення й можливе виправлення помилки. У коді Хемінга цей процес ґрунтується на обчисленні синдрому – векторного результату перевірки паритетних рівнянь, який дозволяє локалізувати збій. Основна перевага коду Хемінга полягає в тому, що кожен можливий однорозрядний збій має свій унікальний синдром, що збігається з двійковим представленням номера позиції, де сталася помилка. Завдяки цьому підхід до корекції помилок стає не лише ефективним, а й обчислювально простим.

Процес декодування починається з прийняття кодового слова, яке може містити одну спотворену позицію. Далі обчислюється синдром множенням прийнятого вектора на транспоновану перевірочну матрицю. У випадку, коли вектор не містить помилок, результат множення дорівнює нульовому вектору. Якщо ж помилка є, синдром буде відмінним від нуля й відображатиме позицію збою в двійковому вигляді. Наприклад, якщо помилка сталася в шостому бітові, синдром матиме вигляд 110, що відповідає числу 6. Така властивість – однозначна відповідність синдрому номеру помилки – є наслідком специфічної побудови перевірочної матриці, стовпці якої збігаються з двійковими номерами позицій бітів кодового слова.

Після обчислення синдрому наступним кроком є ідентифікація помилкової позиції. Це здійснюється шляхом аналізу синдрому та його інтерпретації як двійкового числа. Якщо синдром дорівнює, скажімо, 101, тобто 5 у десятковій системі, то п’ятий біт вважається помилковим. Відповідно, саме цей біт інвертується – якщо він дорівнював 1, змінюється на 0, і навпаки. Таким чином, виконується виправлення єдиної помилки в кодовому слові, що дозволяє точно відновити передане повідомлення.

Що важливо, процедура корекції не вимагає повторної передачі або взаємодії з відправником – вона є повністю самодостатньою на приймальному боці. Це значно підвищує надійність та ефективність систем зв’язку, особливо в умовах, коли зворотній канал або відсутній, або є ненадійним. Після

виправлення біта правильне кодове слово розкладається на складові, і з нього вилучаються інформаційні біти, які й становлять відновлене повідомлення.

Код Хемінга не лише дозволяє виявляти і виправляти одиничні помилки, але й здатен виявляти наявність більшої кількості збоїв, хоча без можливості їх корекції. Якщо у кодовому слові сталися дві або більше помилок, обчислений синдром не відповідатиме жодній допустимій одиночній позиції, а тому не дозволить коректно локалізувати проблему. Це відкриває можливість розширення коду шляхом введення додаткових механізмів виявлення, зокрема біта загального паритету. Але навіть у своїй класичній формі код Хемінга демонструє вражаючу здатність підтримувати цілісність даних при передачі, що й обумовлює його широке застосування в системах з обмеженими обчислювальними ресурсами. У наступному розділі буде розглянуто приклад повного проходження інформації через канал із помилкою – від передавання кодового слова до корекції похибки та відновлення початкового повідомлення.

На завершення розгляду коду Хемінга доцільно зосередитися на його можливостях, що виходять за межі класичного варіанту, а також на сферах практичного застосування, де ці коди демонструють свою ефективність. Варто зазначити, що хоча класичний код Хемінга здатен лише до виправлення одиничних помилок, у деяких випадках цього може бути недостатньо. Для таких задач було запропоновано розширену модифікацію, у якій додається ще один контрольний біт, що відповідає за загальний паритет усього кодового слова. Цей додатковий елемент не збільшує здатність коду до виправлення, однак дозволяє виявити наявність двох одночасних помилок, тим самим значно підвищуючи надійність контролю. Мінімальна відстань такого розширеного коду дорівнює чотирьом, що дає змогу не лише коригувати одну помилку, а й ідентифікувати випадки, коли передача була порушена сильніше, ніж дозволяє базова модель.

З математичного погляду це означає, що простір можливих кодових слів розширюється, але при цьому залишаються збереженими властивості систематичності та алгебраїчної побудови. У доданому паритетному біті

зберігається інформація про те, чи парна кількість одиниць у кодовому слові. Якщо при прийомі синдром вказує на відсутність помилок, а паритетний біт виявляється хибним, це сигналізує про наявність двох помилок, які не можуть бути виправлені, але принаймні будуть виявлені. Така особливість має велике значення у відповідальних системах, де неприйнятно неконтрольовано втратити чи спотворити дані.

Значущість коду Хемінга проявляється також у схемотехнічних реалізаціях. Його структура дозволяє реалізувати апаратний енкодер та декодер, які виконують відповідні логічні операції в реальному часі. На схемному рівні це реалізується через каскади елементів XOR, що визначають значення контрольних бітів під час кодування та виконують перевірку під час декодування. Синхронізація всіх цих операцій забезпечується лінійністю коду і відсутністю потреби у складних циклічних обчисленнях, що критично важливо для вбудованих систем з обмеженими обчислювальними можливостями. Крім того, структура кодового слова з фіксованими позиціями контрольних і інформаційних бітів дозволяє спростити логіку маршрутизації сигналів.

Код Хемінга також ефективно масштабується на довші повідомлення. Для кожного значення параметра r можна побудувати відповідний код довжини $n = 2^r - 1$, що підтримує $k = 2^r - 1 - r$ інформаційних бітів. Таким чином, використовуючи, наприклад, $r = 4$, отримуємо код $(15, 11)$, що дозволяє працювати вже з 11-бітовими повідомленнями, зберігаючи здатність до виправлення однієї помилки. Зростання r забезпечує розширення потужності коду без зміни базових принципів побудови. Ця властивість є надзвичайно корисною в телекомунікаційних протоколах, де передається велика кількість блоків фіксованої довжини, і важливо зберігати уніфіковану логіку контролю.

Таким чином, код Хемінга, попри свою простоту, є гнучким і функціональним інструментом для підвищення надійності передачі інформації. Його здатність до розширення, апаратна реалізованість та ефективна алгебраїчна структура роблять його актуальним не лише у навчальних прикладах, а й у промислових системах, комп’ютерній техніці, цифрових

каналах зв’язку та зберігання даних. Його універсальність дозволяє адаптувати код до широкого спектру завдань без втрати базових переваг, що робить його одним із найвідоміших і найпрактичніших прикладів завадостійкого кодування.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ

Реалізована програма виконує процес кодування довільного текстового повідомлення на основі алгоритму Хаффмана, який передбачає побудову бінарного дерева з метою ефективного стискання даних. У цьому підрозділі розглянуто детальну послідовність дій, які виконує програма, а також фрагменти коду, що реалізують кожен етап.

На початковому етапі програма ініціює введення повідомлення, яке користувач бажає закодувати. Вхідні дані передаються у вигляді рядка символів. Перед обробкою здійснюється перевірка на наявність вмісту. Якщо рядок порожній, виконання припиняється, а користувач отримує відповідне повідомлення. На рисунку 2 показано реалізацію цього процесу.

```
def main():
    # отримання вхідного тексту від користувача
    text = input("Введіть рядок для кодування методом Хаффмана: ").strip()

    if not text:
        print("Ви не ввели рядок.")
        return
```

Рисунок 2

Після успішного введення повідомлення здійснюється побудова таблиці частот входження кожного символу. Цей процес реалізовано за допомогою словника, де кожен ключ відповідає окремому символу, а значення – кількості його появ у вхідному рядку. Рисунок 3 ілюструє процес обчислення частот для кожного символу в тексті.

```
def build_huffman_tree(text):
    # обчислення частоти кожного символу
    freq_dict = {}
    for char in text:
        freq_dict[char] = freq_dict.get(char, 0) + 1
```

Рисунок 3

На основі отриманих частот створюються вузли дерева Хаффмана. Кожен вузол зберігає інформацію про символ, його частоту, а також посилання на лівого та правого нащадка. Структура вузлів задається за допомогою окремого класу NodeTree. Його структура представлена на рисунку 4.

Рисунок 4

```
class NodeTree:
    def __init__(self, symbol=None, freq=None, left=None, right=None):
        self.symbol = symbol  # символ
        self.freq = freq      # частота символу
        self.left = left      # ліва гілка
        self.right = right    # права гілка
```

Наступним етапом є побудова самого дерева. Для цього програма послідовно об'єднує два вузли з найменшими частотами в один спільний вузол, після чого повертає його у список для подальшої обробки. Такий підхід гарантує мінімізацію середньої довжини закодованого повідомлення. Процес триває до моменту, поки у списку не залишиться лише один вузол – корінь дерева. Фрагмент коду, що реалізує побудову дерева, показано на рисунку 5.

```
# побудова дерева шляхом об'єднання найменш частих вузлів
while len(nodes) > 1:
    # сортування вузлів за зростанням частоти
    nodes = sorted(nodes, key=lambda x: x.freq)
    # вибір двох вузлів з найменшою частотою
    left = nodes.pop(0)
    right = nodes.pop(0)
    # створення нового об'єднаного вузла
    merged = NodeTree(freq=left.freq + right.freq, left=left, right=right)
    nodes.append(merged)
```

Рисунок 5

Після побудови дерева здійснюється формування кодів Хаффмана для кожного символу. Це реалізовано шляхом рекурсивного обходу дерева: при русі ліворуч до коду додається цифра «0», а при русі праворуч — «1». Якщо вузол є листовим, тобто містить символ, то згенерований код записується у словник

кодів. На рисунку 6 наведено фрагмент функції, що формує коди для кожного символу.

```
def huffman_code_tree(node, code='', code_dict=None):
    if code_dict is None:
        code_dict = {}
    if node is None:
        return code_dict
    # якщо це лист (кінцевий вузол)
    if node.left is None and node.right is None:
        code_dict[node.symbol] = code or '0'  # обробка випадку з єдиним символом
        return code_dict
    huffman_code_tree(node.left, code + '0', code_dict)
    huffman_code_tree(node.right, code + '1', code_dict)
    return code_dict
```

Рисунок 6

Сформовані коди дозволяють перейти до побудови вихідної таблиці, в якій для кожного символу відображається його частота та відповідна бінарна послідовність. Як видно з рисунка 7, для цього використовується цикл, що проходить по відсортованих ключах словника частот і виводить символи разом з їхніми кодами. Таке подання дає змогу оцінити, наскільки ефективно відбулося кодування з точки зору скорочення обсягу даних.

```
for char in sorted(frequencies.keys()):
    print(f"  {repr(char):<6} |{frequencies[char]:^10}| {huffman_codes[char]}")
```

Рисунок 7

Заключним кроком є процес безпосереднього кодування повідомлення. У цьому етапі, зображеному на рисунку 8, кожен символ вхідного рядка замінюється його відповідним кодом, після чого всі коди об’єднуються у єдину бінарну послідовність. Саме цей рядок і є закодованим повідомленням, яке виводиться у консоль.

```
# кодування повідомлення
encoded = ''.join(huffman_codes[char] for char in text)
print("\nЗакодоване повідомлення:", encoded)
```

Рисунок 8

Таким чином, дана програма повністю реалізує класичний алгоритм Хаффмана – від аналізу вхідних даних до отримання стисненого результату. Завдяки побудові оптимального дерева кодування досягається зменшення розміру інформації без втрати її змісту, що є особливо цінним для задач збереження чи передавання текстових даних.

Щоб перевірити, чи програма правильно кодує повідомлення за методом Хаффмана, були проведені тести на різних прикладах. Програма дозволяє ввести будь-який текст, аналізує, які символи в ньому зустрічаються найчастіше, і на основі цього створює спеціальне дерево. Потім кожному символу призначається свій унікальний бінарний код: ті символи, які зустрічаються частіше, отримують коротші коди, а ті, що рідше – довші. Це дозволяє зробити загальну довжину закодованого повідомлення меншою.

На рисунку 9 показано приклад кодування рядка «aaabbabbbaaa». У цьому рядку два символи: літера a, яка зустрічається 7 разів, і літера b, яка з’являється 5 разів. Оскільки a використовується частіше, їй був призначений коротший код – 1, а b – 0. У результаті кожна літера в рядку замінилася на відповідний біт, і з усього тексту вийшов новий бінарний рядок – 111001000111. Це і є закодоване повідомлення.

```
Введіть рядок для кодування методом Хаффмана: aaabbabbbaaa

 Символ | Частота   | Код Хаффмана
---------------------------------
  'a'    |    7      | 1
  'b'    |    5      | 0

Закодоване повідомлення: 111001000111
```

Рисунок 9

Другий приклад – це рядок «Huffman code». Він більш складний, бо містить багато різних символів, і кожен із них зустрічається лише один раз. Як показано на рисунку 10, програма також працює коректно: для кожного символу

створено свій унікальний код. Наприклад, H отримав код 1010, u – 1011, пробіл – 1111, a – 1101 і так далі. Хоча всі символи мають однакову частоту, програма все одно змогла побудувати правильне дерево та призначити відповідні коди. В результаті утворився бінарний рядок, який представляє закодоване повідомлення:

10101011100110011001111011110111000001010011

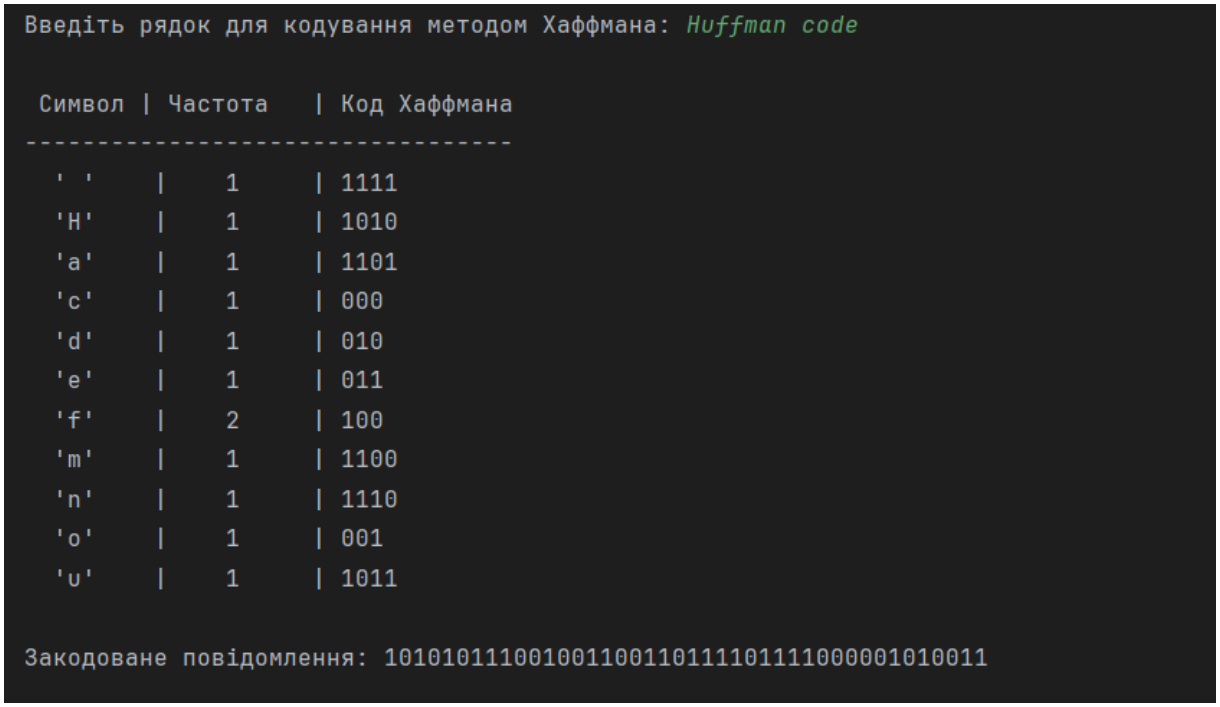

Рисунок 10

Ці приклади підтверджують, що алгоритм працює правильно як у випадках з повторюваними символами, так і тоді, коли символи різні. У кожному випадку програма стискає початковий текст, перетворюючи його в набір нулів і одиниць, що займає менше місця. Це показує, що метод Хаффмана добре підходить для ефективного кодування текстової інформації.

ВИСНОВОК

У дипломній роботі були розглянуті основні поняття теорії кодування. Основну увагу зосереджено на стисканні даних та захисті інформації від помилок. Вивчено алфавітне подання повідомлень, префіксні коди, а також алгоритми Шеннона-Фано і Хаффмана, які дозволяють зменшити обсяг переданої інформації без втрати змісту.

Також проаналізовано механізми, що дозволяють виявляти та виправляти помилки, які виникають у процесі передавання даних. Для цього розглянуто основи побудови завадостійких кодів, зокрема код Хемінга як приклад ефективного лінійного блочного коду.

Під час теоретичного аналізу описано значення таких характеристик, як ентропія, надлишковість та середня довжина коду. Вони визначають ефективність алгоритмів стискання і дають змогу обирати коди, які зменшують розмір повідомлень.

Алгоритм Хаффмана показав свою перевагу в тому, що створює коротші коди для частіших символів. Це дозволяє зменшити загальну кількість бітів у закодованому тексті. У порівнянні з ним, алгоритм Шеннона–Фано працює простіше, але іноді дає менш оптимальні результати.

Окрему увагу приділено дослідженню помилок, які виникають у каналах зв’язку. Було описано типи помилок – одиничні, групові, симетричні та асиметричні. Це дозволило зрозуміти, які саме коди доцільно використовувати в конкретних умовах, щоб забезпечити надійність передачі.

Розділ, присвячений коду Хемінга, показав, як можна виявити і виправити помилку без повторної передачі повідомлення. Такі коди активно застосовуються у цифровій техніці, оскільки легко реалізуються та добре працюють у практичних задачах.

У практичній частині роботи було створено програму, яка реалізує алгоритм Хаффмана. Вона автоматично підраховує частоти символів, будує дерево кодування та стискає повідомлення. Також реалізовано зворотний

процес – декодування. Отримані результати підтвердили правильність теоретичних положень.

Окрім класичних методів, теорія кодування включає й інші цікаві напрямки. Наприклад, існують адаптивні та статистичні методи, які працюють ефективніше за умови змінної структури даних[11]. Такі методи використовуються у складніших системах обробки інформації. Їх вивчення може бути корисним для більш глибокого розуміння теми та розширення знань у цій галузі.

Кодування – це не лише математична теорія. Це важлива складова сучасних цифрових технологій. Завдяки поєднанню теоретичних основ і практичної реалізації вдалося повністю охопити тему та показати її цінність у реальних умовах.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Прокопишин І. А., Рикалюк Р. Є., Чекурін В. Ф., Червінка К. А. Основи теорії інформації та кодування : навч. посібник. — Львів : ЛНУ ім. Івана Франка, 2023. — 156 с.
2. Жураковський Ю. П., Полторак В. П. Теорія інформації та кодування : підручник. — Київ : Вища школа, 2001. — 255 с. : іл. — ISBN 966-642-031-7.
3. Коваленко А. Є. Теорія інформації та кодування : курс лекцій [Електронний ресурс]. — Київ : НТУУ «КПІ ім. Ігоря Сікорського», 2020. — 248 с. URL: https://ela.kpi.ua/bitstream/123456789/41907/1/Kovalenko_AE_TIK_KursLecY20.pdf (дата звернення: 30.04.2025).
4. Майданюк В. П. Кодування та захист інформації : навч. посібник. — Вінниця : ВНТУ, 2009. — 164 с.
5. Lecture 3 Handout [Електронний ресурс]. — Lisboa : Instituto Superior Técnico, 2024. URL: https://fenix.tecnico.ulisboa.pt/downloadFile/563568428875774/Lecture_3_handout.pdf (дата звернення: 30.04.2025).
6. Optimal Codes [Електронний ресурс]. — Linköping : Linköping University, 2024. URL: https://icg.isy.liu.se/courses/TSBK08/lect3.pdf (дата звернення: 30.04.2025).
7. Lin S., Costello D. J. Error Control Coding: Fundamentals and Applications. — Englewood Cliffs (N.J.) : Prentice-Hall, 1983. — 624 с. — ISBN 0-13-283796-X.
8. Huffman W. C., Pless V. Fundamentals of Error-Correcting Codes. — Cambridge : Cambridge University Press, 2003. — 665 с.
9. Wolf J. K. Error Correction [Електронний ресурс]. — San Diego : University of California, 2005. URL: https://acsweb.ucsd.edu/~afazelic/ece154c/ErrorCorrection-JackWolf.pdf (дата звернення: 30.04.2025).

10. Lecture 4 Handout [Електронний ресурс]. — Lisboa : Instituto Superior Técnico, 2024. URL: https://fenix.tecnico.ulisboa.pt/downloadFile/563568428818834/Lecture_4_handout.pdf (дата звернення: 30.04.2025).
11. OCW Delft. Huffman Codes Lecture Notes [Електронний ресурс]. — Delft : TU Delft, 2023. URL: https://ocw.tudelft.nl/wp-content/uploads/Algoritmiek_Huffman_Codes.pdf (дата звернення: 30.04.2025).

ДОДАТКИ

```
2 usages
class NodeTree:
    def __init__(self, symbol=None, freq=None, left=None, right=None):
        self.symbol = symbol  # символ
        self.freq = freq      # частота символу
        self.left = left      # ліва гілка
        self.right = right    # права гілка

    def __lt__(self, other):  # дозволяє порівнювати вузли за частотою
        return self.freq < other.freq
# рекурсивна функція для побудови таблиці кодів Хаффмана
3 usages
def huffman_code_tree(node, code='', code_dict=None):
    if code_dict is None:
        code_dict = {}
    if node is None:
        return code_dict
    # якщо це лист (кінцевий вузол)
    if node.left is None and node.right is None:
        code_dict[node.symbol] = code or '0'  # обробка випадку з єдиним символом
        return code_dict
    huffman_code_tree(node.left, code + '0', code_dict)
    huffman_code_tree(node.right, code + '1', code_dict)
    return code_dict
# побудова дерева Хаффмана
1 usage
def build_huffman_tree(text):
    # обчислення частоти кожного символу
    freq_dict = {}
    for char in text:
```

Додаток А

```
    for char in text:
        freq_dict[char] = freq_dict.get(char, 0) + 1
    # створення списку вузлів
    nodes = [NodeTree(symbol=char, freq=freq) for char, freq in freq_dict.items()]
    # побудова дерева шляхом об'єднання найменш частих вузлів
    while len(nodes) > 1:
        # сортування вузлів за зростанням частоти
        nodes = sorted(nodes, key=lambda x: x.freq)
        # вибір двох вузлів з найменшою частотою
        left = nodes.pop(0)
        right = nodes.pop(0)
        # створення нового об'єднаного вузла
        merged = NodeTree(freq=left.freq + right.freq, left=left, right=right)
        nodes.append(merged)
    # повертаємо корінь дерева та словник частот
    return nodes[0], freq_dict
1 usage
def main():
    # отримання вхідного тексту від користувача
    text = input("Введіть рядок для кодування методом Хаффмана: ").strip()

    if not text:
        print("Ви не ввели рядок.")
        return
    # побудова дерева Хаффмана та обчислення частот символів
    root, frequencies = build_huffman_tree(text)
    # генерація кодів Хаффмана
    huffman_codes = huffman_code_tree(root)
    # виведення таблиці символів, частот і кодів
    print("\n Символ | Частота   | Код Хаффмана")
uffman_tree()
```

Додаток Б

```
    # генерація кодів Хаффмана
    huffman_codes = huffman_code_tree(root)
    # виведення таблиці символів, частот і кодів
    print("\n Символ | Частота   | Код Хаффмана")
    print("----------------------------------")
    for char in sorted(frequencies.keys()):
        print(f"  {repr(char):<6} |{frequencies[char]:^10}| {huffman_codes[char]}")
    # кодування повідомлення
    encoded = ''.join(huffman_codes[char] for char in text)
    print("\nЗакодоване повідомлення:", encoded)
if __name__ == "__main__":
    main()
```

Додаток В