Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: **Розроблення методики кодування інформації в smart-системах**

Виконав: студент VI курсу, групи КН-61м
спеціальності
122 – "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Куцик О.М.
(прізвище та ініціали)
Керівник Процах Н.П.
(прізвище та ініціали)
Рецензент Кінаш Ю.Є.
(прізвище та ініціали)

Львів – 2024 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Борецька І. Б.
"05" січня 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Куцик Олег Михайлович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розроблення методики кодування інформації в smart-системах**
керівник роботи, Процах Наталія Петрівна, д.т.н, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 13.02.2023 року № С-49
2. Термін подання студентом роботи 05. 01. 2024 р.
3. Вихідні дані до роботи:
- провести огляд алгоритмів завадостійкого кодування;
- дослідити математичну модель завадостійкого кодування;
- розробити програмний продукт з інтуїтивним інтерфейсом;
- провести тестування роботи розробленого програмного продукту.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
Висновки
5. Перелік графічного матеріалу:
системний аналіз, розробка та відображення алгоритму роботи завадостійкого кодування, структура програмного рішення, експериментальна частина)
Додатки. Коди розроблених програм кодування та декодування
6. Дата видачі завдання 15 лютого 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання	15.02.2023	виконано
2	Огляд літературних джерел	30.04.2023	виконано
2	Розділ 1. Стан проблемної області	31.07.2023	виконано
3	Розділ 2. Інформаційне забезпечення	30.08..2023	виконано
4	Розділ 3. Математичне забезпечення	29.09.2023	виконано
5	Розділ 4. Програмне забезпечення	31.10.2023	виконано
7	Розділ 5. Розробка стартап-проєкту	30.11.2023	виконано
8	Оформлення пояснювальної записки	29.12.2023	виконано
9	Подання готової роботи	05.01.2024	виконано

Студент _____ Куцик О.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Процах Н.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 59 сторінок пояснювальної записки, 13 рисунків, 6 таблиць, 1 додаток, 20 літературних джерел.

Розроблені методи побудови завадостійких кодів за допомогою багатопозиційних комбінаторних структур типу ідеальних кільцевих в'язанок для створення систем кодування, які виявляють та виправляють помилки, з поліпшеними якісними показниками за потужністю та завадостійкістю.

Розроблена програмна реалізація для синтезу завадостійких кодів на основі ідеальних кільцевих в'язанок, а також представлена візуалізації отриманих результатів.

Ключові слова: *завадостійке кодування, коригуючі коди, ідеальна кільцева в'язанка*.

ANNOTATION

The master's thesis contains 59 pages of explanatory note, 13 figures, 6 tables, 1 appendix, 20 literary sources.

Methods for constructing jamming-resistant codes using multi-position combinatorial structures of the type of perfect ring-blankets are developed to create error-detecting and error-correcting coding systems with improved power and jamming performance.

A software implementation for the synthesis of interference-resistant codes on the basis of ideal circular knittings is developed, and visualizations of the obtained results are also presented.

Keywords: *correction codes, ideal ring bundle, interference-resistant coding*.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне та алгоритмічне забезпечення завадостійкого перетворення інформації для виправлення помилок, а саме:

1. провести попередній аналіз існуючих завадостійких кодів для виправлення помилок;
2. визначити математичну модель завадостійких кодів корекції помилок;
3. визначити кількість помилок, виявлених та виправлених завадостійким кодом;
4. розробити алгоритм синтезу завадостійкого коду, що виявляє та виправляє помилки;
5. інтерпретувати отримані результати;
6. розробити програмне забезпечення для представлення результатів роботи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ОДИНИЦЬ І ТЕРМІНІВ

ІКВ	ідеальна кільцева в’язанка
ПЗ	програмне забезпечення
ЦК	циклічний код
BCH	Bose-Chowdhury-Hocquenghem
CRC	Cyclic Redundancy Check

ВСТУП

Актуальність. На даний момент досліджуємо комбінаторні властивості ідеальної кільцевої в’язанки з метою побудови багатопозиційних кодів для створення інформаційних технологій та обчислювальних пристроїв з високими показниками якості – такими як забезпечення достовірності інформації, підвищення надійності, функціональної безпеки та живучості інформації та систем управління інформацією.

Важливо створювати системи кодування з покращеними якісними показниками за потужністю та швидкістю виявлення та виправлення помилок у повідомленнях, що надсилаються по каналах зв’язку.

Мета і задачі дослідження. Метою магістерської роботи є синтез та дослідження оптимізованих циклічних кодів з використанням ідеальних кільцевих в’язанок.

Для досягнення поставлених цілей в роботі вирішуються наступні завдання:

- переглянуто існуючі завадостійкі коди та завадостійке кодування;
- проведено системний аналіз циклічних кодів;
- проаналізовано метод побудови багатопозиційного помилково стійкого коду на основі ідеальних кільцевих в’язанок;
- проаналізовано методи підвищення потужності завадостійких багатопозиційних кодів;
- розробка стартап-проекту;

Наведені задачі розв’язуються в п’яти розділах магістерської роботи. У першому розділі розглядаються характеристики циклічних кодів з використанням ідеальних кільцевих в’язанок. Опис існуючих циклічних кодів розглядається у другому розділі магістерської роботи.

У третьому розділі було проведено системний аналіз двійкових блокових кодів, проаналізовано коди Бозе-Чоудхурі-Хоккенгама. Розглянуто та проаналізовано метод побудови багатопозиційного завадостійкого коду на

основі ідеальних кільцевих в’язанок та методи підвищення потужності багатопозиційних завадостійких кодів.

Програмну реалізацію та опис програми можна знайти в четвертому розділі.

У п'ятій частині розробляється проект стартапу.

Наукова новизна отриманих результатів. У цій роботі було розроблено програмне забезпечення, яке дозволяє на прикладі зображення візуально побачити, як виглядає зображення до і після шуму. Аналізуючи результати, можна зробити висновок, що вихідне зображення легко розпізнається, що забезпечує автентичність і достовірність переданої інформації.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд основних понять і параметрів коду

Розроблене програмне забезпечення дозволяє виявляти та виправляти помилки. На прикладі зображення можна наочно побачити, як програма виправляє помилки. Це показано та описано більш детально в розділі 4.

Цей додаток можна вдосконалити та ефективно використовувати в інформаційних технологіях. Програма забезпечує якісні показники, а саме забезпечення достовірності інформації, підвищення достовірності при декодуванні.

Багатопозиційні оптимізовані коди дозволяють виправляти більшу кількість помилок однаковою кількістю бітів порівнюваних кодів, майже прямо пропорційних кількості бітів, але не перевищуючи 25% від загальної кількості бітів кодової комбінації, зберігаючи при цьому постійне значення потужності коду корекції.

Після завершення програми отримані дані аналізуються та робляться висновки. Як уже було сказано вище, якщо програма буде вдосконалена в майбутньому, її можна буде розробити та використовувати як систему клієнт-сервер.

Основною перевагою багатопозиційного коду з оптимізованими параметрами є можливість виправлення до 25% помилок при стабільному рівні ефективності коригуючої здатності коду з будь-яким великим числом його розрядів, що відрізняє цей код від стандартного. коди, в тому числі коди BCH [16].

В якості технічного рішення для використання замовником підійде настільний або портативний комп'ютер з такими мінімальними характеристиками:

- одно і багатопроцесорні рішення у яких частота ядра перевищує 2000 Мгц;
- оперативна пам'ять DDR4 не менше 4096 MB;

Якщо аналітична обробка буде проводитися на конкретному комп'ютері, то має сенс встановити на цей комп'ютер 8 Гб оперативної пам'яті, більш продуктивний багатоядерний процесор і більш потужну материнську плату, оскільки під час аналітичної обробки максимально використовується апаратний потенціал і сам час обробки може істотно відрізнятися.

Вищенаведені вимоги є оптимальними для функціонування програмного забезпечення, покращення будь-якої з вимог не призведе до значного підвищення ефективності роботи.

1.2. Класифікація завадостійких кодів

Усі коди виявлення та виправлення помилок можна розділити на два класи: блокові та безперервні.

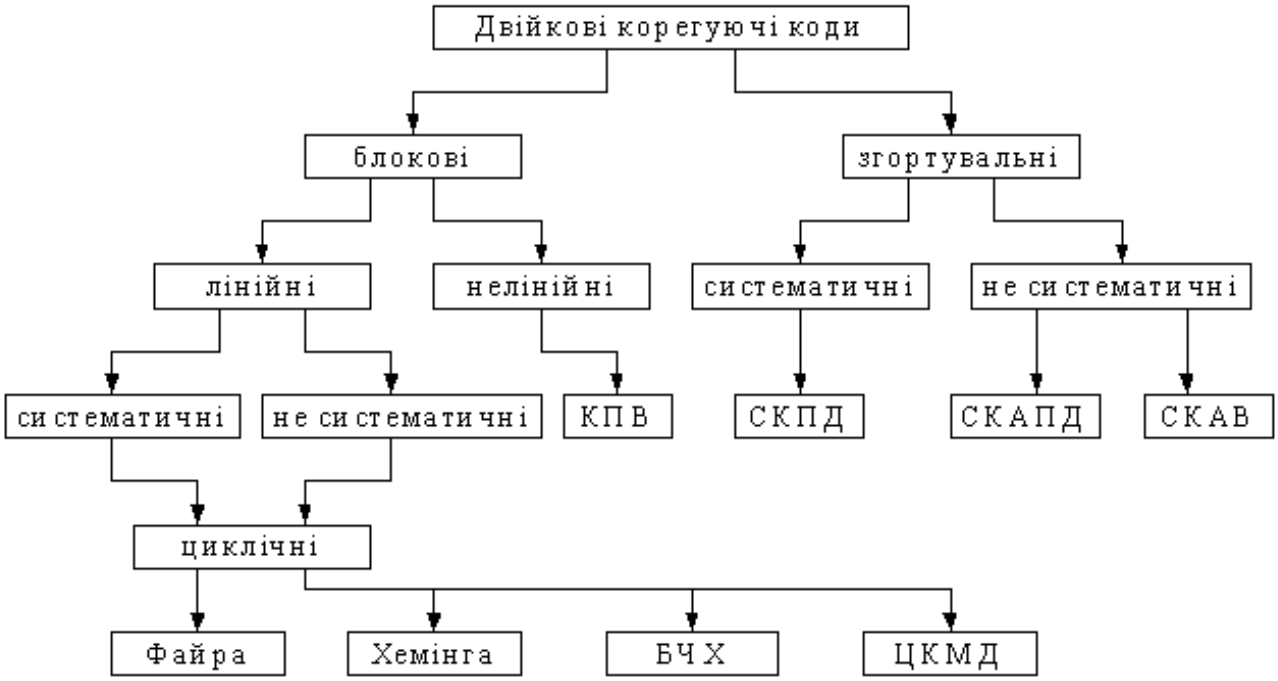

Рис. 1.1. Класифікація корегуючих кодів

Роздільні коди забезпечують чітке розрізнення інформаційних і перевірочних символів.

Нерозривні коди не допускають такої можливості.

Лінійні коди – Це коди, де, якщо знаєте дві дозволені комбінації кодів, можете отримати третю дозволену комбінацію кодів за допомогою лінійної операції.

Однією з таких лінійних операцій є підсумовування за модулем 2.

Сьогодні розроблено десятки кодів, які теоретично можуть виявляти будь-яку кількість помилок.

За наявності такого різноманіття завадостійких кодів чітке розділення їх на групи на основі взаємно непересічних характеристик є нереальним завданням.

1.3. Огляд завадостійкого кодування

Що таке канали зв'язку?

Канали зв’язку стосуються середовища, через яке передаємо інформацію. Кожен канал зв'язку має свій шумовий фактор, який певною мірою спотворює інформацію та загрожує її достовірності.

До цих пір людство не змогло усунути джерела шуму, але знайшло обхідні шляхи.

Завдання полягало в тому, щоб знайти спосіб кодування, який, пройшовши через зашумлений канал зв'язку, міг би частково або повністю виправити спотворені елементи повідомлення.

Такі способи знайдено. Це призвело до завадостійкого кодування [19,14].

Особливістю кодів виявлення помилок є те, що кодові комбінації, що містяться в цих кодах, відрізняються на кодову відстань не менше ніж $d_{min} = 2$.

Висновок до розділу 1

Такі коди можна умовно поділити на дві групи: коди, в яких використовуються всі комбінації, але до кожної з них за встановленим правилом додаються *r* елементи управління; коди, створені шляхом зменшення кількості дозволених комбінацій.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Огляд ідеальних кільцевих в'язанок(ІКВ)

Ідеальною кільцевою в'язанкою (ІКВ) [22, 15] називається послідовність $K_N=(k_1,k_2,...,k_N)$ чисел, на якій всі можливі кільцеві суми вичерпують значення чисел натурального ряду $1,2,...,S_N$, де:

$$S_N=N(N-1)+1. \tag{2.1}$$

Наведемо приклад простої ІКВ шостого $(N=6)$ порядку. Згідно (2.1) маємо $S_6=6(6-1)+1=31$. Можна пересвідчитися, що в табл. 2.1 кільцевих сум, цієї ІКВ, містяться всі числа натурального ряду від 1 до 31.

Таблиця 2.1

Кільцеві суми для ІКВ шостого порядку $(1, 3, 2, 7, 8, 10)$

p_j	q_j					
	1	2	3	4	5	6
1	1	4	6	13	21	31
2	31	3	5	12	20	30
3	28	31	2	9	17	27
4	26	29	31	7	15	25
5	19	22	24	31	8	18
6	11	14	16	23	31	10

Отже, послідовність $(1,3,2,7,8,10)$ утворює одне із ІКВ 6-го порядку.

Окрім простих ідеальних кільцевих шаблонів, існує багато ідеальних кільцевих шаблонів, які є послідовностями N чисел, які містять повторювані числа в натуральному ряду.

Багато ідеальних в'язанок утворюються послідовністю N цілих чисел $K_N=(k_1,k_2,...,k_i,...,k_N)$, на якій кільцеві суми набувають значення чисел

натурального ряду , а кожне з чисел $1, 2, ..., S_N^R = S_N$-1 є значенням R різних кільцевих сум. Між кількістю чисел N, кратністю R та сумою S_N всіх чисел R -кратного ідеального кільця існує співвідношення:

$$S_N = \frac{N(N-1)}{R} + 1. \qquad (2.2)$$

Наведемо приклад 3-кратної $(R=3)$ ІКВ шостого $(N=6)$ порядку (табл. 2.2).

Таблиця 2.2

Суми ІКВ шостого порядку третьої кратності $(1, 1, 2, 1, 2, 4)$

p_j ¤ ¤	¤	¤	q_j ¤	¤	¤	¤
	1¤	2¤	3¤	4¤	5¤	6¤
1¤	1¤	2¤	4¤	5¤	7¤	11¤
2¤	11¤	1¤	3¤	4¤	6¤	10¤
3¤	10¤	11¤	2¤	3¤	5¤	9¤
4¤	8¤	9¤	11¤	1¤	3¤	7¤
5¤	7¤	8¤	10¤	11¤	2¤	6¤
6¤	5¤	6¤	8¤	9¤	11¤	4¤

За формулою (2.2) маємо $S_6 = \frac{6(6-1)}{3} + 1 = 11$. Можна пересвідчитися, що в табл. 2.2 кільцеві суми відповідають ІКВ.

По аналогії з ІКВ, які генерують всі додатні числа в діапазоні від 1 до S_N, використовуючи вдосконалені алгоритми побудови нееквідистантних конфігурацій, можна знайти нові ІКВ з від'ємними елементами, що дозволяє отримати всі значення в діапазоні від $-\frac{S_N - 1}{2}$ до $-\frac{S_N - 1}{2}$, де S_N обчислюється за формулою (2.8).

Наведемо приклад існування таких ІКВ четвертого $(N=4)$ порядку кратності $(R=1)$ (табл. 2.3).

Таблиця 2.3

Кільцеві суми для ІКВ з від'ємними елементами четвертого порядку першої кратності $(-1,\ -3,\ -2,\ 6)$

p_j	q_j			
	1	2	3	4
1	-1	-4	-6	0
2	0	-3	-5	1
3	3	0	-2	4
4	5	2	0	6

Всього згаданих ІКВ з такими параметрами існує чотири варіанти: $(1, 3, 2, -6)$; $(-1, -3, -2, 6)$; $(1, -4, -2, 5)$; $(-1, 4, 2, -5)$. За формулою (2.12) маємо $S_4 = \frac{4(4-1)}{1} + 1 = 13$. Можливо, у таблиці 2.5 є конкретні підсумкові суми, наприклад ІКВ: $(-1, -3, -2, 6)$, містить усі числа від -6 до 6, і кожне число є значенням кільцевої суми з різними цифровими кодами.

ІСІ з негативними елементами можна використовувати в параметричних моделях технічних комбінаторних систем, де значення параметрів можуть мати як позитивні, так і негативні значення.

2.2. Аналіз циклічних кодів

Методи комбінаторної оптимізації широко використовуються в інформаційних технологіях для кодування, обробки і передачі даних, в інформаційних технологіях, вимірюваннях і обчисленнях, радіотехніці, зв'язку та суміжних галузях науки і техніки.

Приклади постановки таких завдань: проектування перешкодостійких систем кодування, розробка ефективних технологій захисту цінних паперів від

несанкціонованого доступу, удосконалення компонентів комп'ютерних систем, проектування радіосистем високої роздільної здатності.

Тому актуальним є створення систем кодування з підвищеними якісними показниками за потужністю та ефективністю виявлення та виправлення помилок у повідомленнях, що передаються по каналах зв’язку.

Дослідження комбінаторних властивостей ідеальних кільцевих в’язанок для побудови багатопозиційних кодів для створення пристроїв інформаційної техніки та комп’ютерної техніки з високими показниками якості – такими як забезпечення надійності інформації, підвищення надійності, функціональної безпеки та живучості інформації та систем управління інформацією.

Ці коди також широко використовуються для захисту інформації від помилок [2, 21]. Перевірними елементами, що визначає ця матриця, будуть

$$b_1 = a_1 \oplus a_3 \oplus a_4;\ b_2 = a_1 \oplus a_2 \oplus a_3;\ b_3 = a_2 \oplus a_3 \oplus a_4$$

Інший матричний метод побудови циклічного коду ґрунтується на твірному поліномі *P(x).* За цим методом при побудові твірної матриці $G_ц$ як її рядки беруть k лінійно незалежних комбінацій, що відповідають поліномам $x^0P(x), x^1P(x), \ldots, x^{k-1}P(x)$ [1]:

$$G_ц = \begin{bmatrix} x^0P(x) \\ x^1P(x) \\ \ldots\ldots\ldots \\ x^{k-1}P(x) \end{bmatrix} \qquad (2.3)$$

Додаванням по два, три і т. д. до k рядків за модулем 2 дістають усі $2^k – k – 1$ ненульові комбінації циклічного коду, що залишилися.

Так, якщо задатися твірним поліномом $P(x) = x^3 + x^2 + 1$, можна побудувати циклічний код (7,4), який виправляє одноразові помилки.

Як $k = 4$ рядки матриці $G_ц$ використовуються поліноми

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

тобто

$$G_{\text{ц}(7,4)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (2.4)$$

З $2^k = 2^4 = 16$ комбінації циклічного коду (7,4), описуваного цією твірною матрицею, перша комбінація - нульова, друга - п'ята - це рядки твірної матриці, а решта 11 є за модулем 2 сумами всіх можливих зв'язків (комбінацій) рядків твірної матриці.

Генерацію матриці (2.4) можна звести до генерації G(7,4) матриці (2.3) лінійного коду систематичної групи, для чого виконуємо кілька простих операцій: переставляємо перший і четвертий рядки, другий і третій, потім до отриманого третього рядка додаємо другий і третій рядки до першого рядка, третій і четвертий рядки до другого рядка, четверті рядки до третього рядка, а четвертий рядок перепишемо без змін.

В результаті матимемо

$$G_{\text{ц}(7,4)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad (2.5)$$

Звідси за аналогією з лінійним систематичним груповим (*n, k)*- за допомогою коду легко перетворити цю матрицю в перевірочну матрицю (3.3) циклічного коду та знайти елементи переносу:

$$b_1 = a_1 \oplus a_3 \oplus a_4;\ b_2 = a_1 \oplus a_2 \oplus a_3;\ b_3 = a_2 \oplus a_3 \oplus a_4$$

Циклічний код можна побудувати також за допомогою перевірного полінома *H(x)* степеню *k,* який дістають діленням двочлена $x^n + 1$ на твірний поліном *P(x),* тобто

$$H(x) = (x^n + 1)/P(x).$$

Так, для циклічного (7,4)-коду, твірний поліном якого $P(x) = x^3 + x + 1$, перевірний поліном має вигляд

$$H(x) = (x^7 + 1)/(x^3 + x + 1) = x^4 + x^2 + x + 1,$$

або у двійковому запису $H(0,1) = 10111$.

Перевірна матриця при цьому будується за аналогією з твірною матрицею (2.3):

$$H_{\text{ц}} = \begin{bmatrix} x_1^0 H(x) \\ x_1^1 H(x) \\ \cdots\cdots\cdots \\ x^{r-1} H(x) \end{bmatrix} \tag{2.6}$$

Для розглядуваного прикладу [$P(x) = x^3 + x + 1$] перевірна матриця має такий вигляд:

$$H_{\text{ц}(7,3)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.7}$$

Щоб зберегти керуючі співвідношення, матрицю (2.7) необхідно канонізувати (з перевіркою одиниць під матрицею), що робиться аналогічно створенню подібних матриць повторюваного циклічного коду:

$$H_{\text{ц}(7,3)} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$
$$a_1 \; a_2 \; a_3 \; a_4 \quad b_1 \; b_2 \; b_3$$

Використовуючи останню матрицю, легко отримати залежності перевірки кодування. Для розглянутого прикладу вони мають вигляд

$$b_1 = a_1 \oplus a_3 \oplus a_4;\; b_2 = a_1 \oplus a_2 \oplus a_3;\; b_3 = a_2 \oplus a_3 \oplus a_4$$

Звідси маємо такі рівняння декодування:

$$a_1 \oplus a_3 \oplus a_4 \oplus b_1 = 0;\; a_1 \oplus a_2 \oplus a_3 \oplus b_2 = 0;\; a_2 \oplus a_3 \oplus a_4 \oplus b_3 = 0.$$

Висновок до розділу 2

Головна ідея методу є в побудові твірного полінома, вид якого визначається двома параметрами: довжиною кодового слова $\boldsymbol{S}$ і максимальною кількістю виправлених помилок $\boldsymbol{t}$. Інші параметри, що беруть участь у побудові, твірного полінома можна визначаються відповідно до спеціальних таблиць та залежностей [2].

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Декодування циклічних кодів

Перевірну матрицю $H_{ц}$ може бути використаний для ідентифікації та виправлення помилок у циклічному коді шляхом визначення синдрому коду (аналогічного коду лінійної систематичної групи). Однак частіше для цього використовують методи, засновані на використанні твірного полінома *P(x).*

У цьому випадку виявлення помилок при декодуванні зводиться до ділення отриманої кодової комбінації на той же породжуючий поліном, що і при кодуванні.

Частина, що залишилася, тобто *синдромом*, коли вона має *r* нульових коефіцієнтів, свідчить про відсутність помилки. Якщо ж остача не нуль, то існує помилка [3].

Розглянемо випадок виправлення однієї помилки (однаково виправляється багато помилок, але навантаження в кілька разів більше). Визначення місця виникнення помилки в циклічному коді засноване на порівнянні результатів деякої послідовно виконуваної операції з синдромом помилки.

Ця операція заснована на тій властивості, що при діленні комбінація приймається з помилкою *F(x)* на поліномі *P(x)* та частина *F(x),* яка дорівнює переданій комбінації *F(x),* ділиться без остачі, а власне остача визначається вектором помилки *E(x):*

$$F'(x) = F(x) \oplus E(x).$$

Оскільки $F(x) = C(x)P(x)$, маємо

$$\frac{F'(x)}{P(x)} = C(x) \oplus \frac{E(x)}{P(x)}.$$

Знаючи заздалегідь або виконуючи послідовно операції ділення векторів помилки [чи циклічно зсунутих комбінацій *F(x),* що дозволяється з урахуванням кількості зсувів] на поліном *P(x),* випадково можна чітко визначити місце помилки [4].

Для виправлення помилок у прийнятій на приймальному боці комбінації *F(x)* циклічного коду, ви можете використовувати різні методи. Проте найпростішим способом реалізації алгоритму визначення похибок є метод гіпотези, суть якого полягає в наступному.

Як згадувалося вище, якщо, наприклад, є одна помилка в кодовій комбінації *F(x)* остача від ділення цієї комбінації на твірний поліном *P(x)* не дорівнює нулю.

Для знаходження місця помилки виконуємо такі операції:

Крок 1. Будуємо гіпотезу про помилку в молодшому розряді комбінації *F(x)*, тобто припускаємо, що вектор помилки $E_1(x) = 1$ ($E_1 = 00...001$). Підсумовуємо $F(x) \oplus E_1(x)$ і ділимо цю суму на поліном *P(x)* із метою підтвердження (в разі залишку нуля) або спростування (в разі ненульової остачі) гіпотези. Якщо остача *R(x)* не дорівнює нулю, то гіпотеза відкидається.

Крок 2. Будуємо припущення про помилку в другому розряді комбінації *F(x),* тобто вектор помилки $E_2(X) = x$ ($E_2 = 00.. .010$). Підсумовуємо $F(x) + E_2(x)$ і ділимо цю суму на поліном *P(x)* із метою підтвердження або спростування цієї гіпотези. При $R(x) \neq 0$ гіпотеза відкидається.

Крок 3. Будуємо послідовно гіпотези про наявність помилки в третьому, четвертому і т. д. розрядах комбінації *F(x)*, для чого відповідні вектори помилок $E_i(x)(i = 3,4,..., n)$ додаємо до комбінації *F(x)* і результат ділимо на поліном *P(x)* до здобуття остачі $R(x) = 0$, тобто до підтвердження гіпотези.

Остача $R(x) = 0$ свідчить про те, що помилку виправлено. Сума прийнятої комбінації *F(x)* з вектором помилки $E_i(x)$, яка дає остачу $R(x) = 0$, відповідає початковій комбінації *F(x)* циклічного коду, яка передається в канал, тобто:

$$F(x) = F(x) \oplus E_1(x).$$

Інший поширений метод виявлення та виправлення помилок у прийнятій на приймальному боці кодовій комбінації $F(x)$ циклічного коду після здобуття остачі $R(x) \neq 0$ відділення прийнятої комбінації на твірний поліном, зводиться до виконання таких процедур:

- підраховується вага w остачі, тобто кількість одиниць в ній. Якщо $w < v_{вп}$, де $v_{вп}$ — кількість помилок, яку дає змогу виправити код, то прийнята комбінація $F(x)$ додається за модулем 2 до остачі. Сума й дає виправлену комбінацію;
- якщо $w > v_{вп}$, то виконується циклічний зсув прийнятої комбінації $F(x)$ на один розряд ліворуч і здобута комбінація знову ділиться на твірний поліном $P(x)$. Якщо при цьому вага здобутої остачі $w \leq v_{вп}$, то циклічно зсунута комбінація додається за модулем 2 до остачі, а потім циклічно зсувається на один розряд праворуч (повертається в попередній стан). Утворена таким чином комбінація вже не містить помилок;
- якщо ж після першого циклічного зсуву й наступного ділення вага остачі, як і раніше, $w > v_{вп}$, то виконуються додаткові циклічні зсуви ліворуч. При цьому після кожного зсуву утворена комбінація ділиться на поліном $P(x)$ і перевіряється вага остачі. Це робиться доти, доки не буде здобуто вагу остачі $w \leq v_{вп}$. Тоді комбінація, утворена внаслідок останнього циклічного зсуву, додається за модулем 2 до остачі від ділення цієї комбінації на поліном $P(x)$, після чого виконується циклічний зсув праворуч на стільки розрядів, на скільки було зсунуто її відносно прийнятої комбінації $F(x)$. Як результат буде здобуто виправлену комбінацію $F(x)$ циклічного коду.

3.2. Укорочені циклічні коди

Циклічні $(n,\ k)$-коди, що містять 2^k комбінацій, називаються *повними*. З кожного повного можна утворити вкорочений (неповний) циклічний $(n - i,\ k - i)$-код (де $i = 1 \ldots k - 1$).

Такий код має 2^{k-i} комбінацій і кодову відстань $d_{\min}$ не меншу, ніж у повного циклічного коду.

Процес укорочення повного циклічного коду полягає у виборі з 2^k його поліномів тільки частини із членами не вище x^{n-i-1}. Ці поліноми мають вигляд

$$V_y(x) = a_{n-i-1}x^{n-i-1} + \ldots + a_1x + a_0.$$

Усі комбінації вкороченого циклічного коду діляться на твірний поліном $P(x)$ повний циклічний код, з якого він був отриманий. Однак циклічне зміщення комбінації короткого коду не завжди призводить до того, що комбінація належить цьому коду, тобто код не має циклічних властивостей. Тому короткі коди не є циклічними і тому часто називаються *псевдоциклічними.*

Твірну матрицю G_y вкороченого циклічного $(n - i,\ k - i)$-коду дістають з матриці $G_ц$ повного циклічного коду виключенням з неї перших i рядків і стовпців, а перевірну матрицю H_y – з матриці $H_ц$ повного коду виключенням перших стовпців.

Так, твірна та перевірна матриці вкороченого циклічного (6,3)-коду можуть бути здобуті з відповідних матриць (3.1) і (3.2) циклічного (7,4)-коду з твірним поліномом $P(x) = x^3 + x^2 + 1$:

$$G_{(6,3)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}; \qquad H_{(6,3)} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Ґрунтуючись на твірній матриці $G_{(6,3)}$, дістаємо такі комбінації вкороченого циклічного коду: 000000, 100011, 010111, 001101, 110100, 101110, 011010, 111001.

Цей код є псевдоциклічним, тому що циклічний зсув ліворуч на один елемент, наприклад комбінації 100011, дає комбінацію 111000, яка не належить до даного коду.

3.3. Аналіз кодів Боуза–Чоудхурі–Хоквінгема

Вони дозволяють виявити і виправити будь-яку кількість помилок. Під час кодування визначається кількість помилок, які необхідно виправити, або мінімальна кодова відстань і загальна кількість n елементів коду Боуза — Чоудхурі — Хоквінгема (БЧХ) [16].

Так, розкладаючи 2^{h-1} на співмножники, знаходимо такі значення n і g

$7 = 2^3 - 1 = 7$; $63 = 2^6 - 1 = 7*3*3 = 21*3$; $511 = 2^9 - 1 = 73*7$;

$15 = 2^4 - 1 = 5*3$; $127 = 2^7 - 1 = 127$; $1023 = 2^{10} - 1 = 31*11*3$;

$31 = 2^5 - 1 = 31$; $255 = 2^8 - 1 = 17*5*3$; $2047 = 2^{11} - 1 = 89*23$.

Звідси випливає, що при $h = 6$ довжина n кодової комбінації може дорівнювати не тільки 63, а й 21 (при $g = 3$).

Кількість перевірних елементів коду визначається виразом

$$r \le \frac{h(d-1)}{2} = [\log_2(n+1)]\frac{d-1}{2} \qquad (3.1)$$

а кількість інформаційних елементів — виразом

$$k \ge \left(2^h - 1\right) - \frac{h(d-1)}{2} \quad \text{або } k = n - r. \qquad (3.2)$$

У табл. 3.1 наведено деякі мінімальні поліноми кодів БЧХ.

Таблиця 3.1

Номер мінімального полінома	Мінімальні поліноми різного степеня l			
	2	3	4	5
$M_1(x)$	$x^2 + x + 1$	$x^3 + x + 1$	$x^4 + x + 1$	$x^5 + x^2 + 1$
$M_3(x)$		$x^3 + x^2 + 1$	$x^4 + x^3 + x^2 + x + 1$	$x^5 + x^4 + x^3 + x^2 + 1$
$M_5(x)$			$x^2 + x + 1$	$x^5 + x^4 + x^2 + x + 1$
$M_7(x)$			$x^4 + x^3 + 1$	$x^5 + x^3 + x^2 + x + 1$
$M_9(x)$				$x^5 + x^4 + x^2 + x + 1$
$M_{11}(x)$				$x^5 + x^4 + x^3 + x + 1$
$M_{13}(x)$				

Продовження таблиці 3.1

Номер мінімального полінома	Мінімальні поліноми різного степеня *l*			
	6	7	8	9
$M_1(x)$	$x^6 + x + 1$	$x^7 + x^3 + 1$	$x^8 + x^4 + x^3 + x^2 + 1$	$x^9 + x^4 + 1$
$M_3(x)$	$x^4 + x^4 + + x^2 + x + 1$	$x^7 + x^3 + + x^2 + x + 1$	$x^8 + x^6 + x^5 + + x^4 + x^2 + x + 1$	$x^9 + x^6 + x^4 + + x^3 + 1$
$M_5(x)$	$x^6 + x^5 + + x^2 + x + 1$	$x^7 + x^4 + x^3 + + x^2 + 1$	$x^8 + x^7 + x^6 + + x^5 + x^4 + x + 1$	$x^9 + x^8 + x^5 + + x^4 + 1$
$M_7(x)$	$x^6 + x^3 + 1$	$x^7 + x^6 + x^5 + + x^4 + x^2 + + x + 1$	$x^8 + x^6 + x^5 + + x^3 + 1$	$x^9 + x^7 + x^4 + + x^3 + 1$
$M_9(x)$	$x^3 + x^2 + 1$	$x^7 + x^5 + x^4 + + x^3 + x^2 + + x + 1$	$x^8 + x^7 + x^5 + + x^4 + x^3 + x^2 + 1$	$x^9 + x^8 + x^4 + + x + 1$
$M_{11}(x)$	$x^6 + x^5 + x^3 + + x^2 + 1$	$x^7 + x^6 + x^4 + + x^2 + 1$	$x^8 + x^7 + x^6 + x^5 + + x^2 + x + 1$	$x^9 + x^5 + x^3 + + x^2 + 1$
$M_{13}(x)$		$x^7 + x + 1$	$x^8 + x^5 + x^3 + + x + 1$	$x^9 + x^6 + x^5 + + x^4 + x^2 + x + 1$

Після підстановки значень *M(x)* набуває вигляду

$$P^8(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) = x^8 + x^7 + x^6 + x^4 + 1 \rightarrow 111010001.$$

Найбільший степінь твірного полінома *P(x)* визначає кількість перевірних елементів у комбінації ($r = 8$), а кількість її інформаційних елементів $k = n - r = 15 - 8 = 7$. Маємо (15, 7)-код БЧХ з $v_{вп} = 2$.

При необхідності кодову матрицю БЧХ можна побудувати за правилами побудови такої матриці для циклічного коду. Тому для прикладу, що розглядається, аналогічно (3.6) і (3.7) породжувальна матриця коду БЧХ матиме вигляд

$$G_{БЧХ(15,7)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Усі подальші процедури кодування виконуються аналогічно методам, описаним вище для циклічних кодів.

У табл. 3.2 наведено параметри деяких кодів БЧХ довжиною до $n = 255$. Параметри, наведені в таблиці, визначені відповідно до наведеної вище методології.

Коди БЧХ [16] характеризують деякі закономірності. По-перше, співвідношення між максимальною кодовою відстанню та числом h може бути подане як

$$d_{max} = 2^{h-1} - 1.$$

Дійсно, для прикладу, що розглядався вище, при $n = 15$ ($h = 4$) $d_{max}=2^{4-1}-1=7$, а кількість інформаційних розрядів, яка може бути використана при обумовлених значеннях h і d_{max}, дорівнює (h +1).

По-друге, кількість кодів, що різняться своєю коректувальною здатністю і мають однакову довжину кодової комбінації ($n = 2^h - 1$), на дві одиниці менша від кількості всіх незвідних поліномів, на які розкладається двочлен $x^{2^h-1}+1$.

Так, для n - 15 дістанемо $h = 4$ та двочлен $(x^{15} + 1)$. Цей двочлен не є найпростішим, тому $h = 4$ буде старшим степенем незвідного полінома, на який розкладається двочлен $x^{15} + 1$. Таким чином, двочлен $x^{15} + 1$ буде розкладатися на незвідні поліноми четвертого степеню та на незвідні поліноми тих степенів, показники яких є дільниками числа 4, тобто 1 і 2:

$$x^{15} + 1 = (x + 1)(x^2 + x + 1)\ (x^4 + x + 1)(x^4 + x^3 + 1)(x^4 + x^3 + x^2 + x + 1).$$

Як випливає з цього розкладу, кількість незвідних поліномів дорівнює п’яти, а кількість циклічних кодів для n = 15 – 5–2 = 3, що підтверджується табл. 3.2.

Таблиця 3.2

N	k	r	d_{min}	Твірний поліном $P(x)$
7	*4*	3	3	13
15	11	4	3	23
	7	8	5	721
	5	10	7	2467
31	26	5	3	45
	21	10	5	3551
	16	15	7	107657
	11	20	11	5423325
	6	25	15	313365047
63	57	6	3	103
	51	12	5	12471
	45	18	7	1701317
	39	24	9	166623567
	36	27	11	1033500423
	3O	33	13	1574641656547
	24	39	15	17323260404441
	18	45	21	1363026512351725

Продовження табл. 3.2

n	k	r	d_{min}	Твірний поліном $P(x)$
127	120	7	3	211
	113	14	5	41567
	106	21	7	11554743
	99	28	9	3447023271
	92	35	11	624730022327
	85	42	13	130704476322273
	78	49	15	26230002166130115
	71	56	19	6255010713253127753
	64	63	21	1206534025570773100045
255	247	8	3	435
	239	16	5	267543
	231	24	7	156720665
	223	32	9	75626641375
	215	40	11	23157564726421
	207	48	13	16176560567636227
	199	56	15	7633031270420722341
	191	64	17	2663470176115333714567
	187	68	19	52755313540001322236351
	179	76	21	226247107173404324163004 55

При декодуванні кодів БЧХ з довжиною комбінацій $n > 15$ можуть виникнути деякі труднощі, пов'язані з великим обсягом обчислень для виявлення та виправлення помилок.

У таких випадках при $k > n/2$, де k-кратність зсуву, рекомендується комбінацію, утворену після k-кратного зсуву і підсумовування з остачею, зсувати не праворуч, а ліворуч на n циклічних кроків.

3.4. Методи підвищення потужності багатопозиційних завадостійких ІКВ-кодів

Відомо, що потужність будь-якого циклічного коду, в тому числі і коду ІКВ, визначається кількістю його розрядів і зростає зі збільшенням довжини кодових комбінацій.

Одним із відомих способів збільшення потужності ІКВ є використання, крім основної циклічної кодової комбінації, комбінацій, у яких символи всіх розрядів змінюються на протилежні.

При цьому міцність коду подвоюється без зниження його коригувальних можливостей завдяки використанню комбінацій з інверсними символами [5]. Інший підхід полягає у використанні багатьох варіантів ІКВ, які створюють повне сімейство, тобто числові послідовності з однаковими параметрами, для побудови коду п, *К* але різний склад елементів [5].

Теоретичні розрахунки і комп'ютерна перевірка запропонованого методу підтвердили доцільність використання такого підходу для збільшення потужності завадостійкого ІКВ-коду.

Алгоритм синтезу ІКВ-коду підвищеної потужності передбачає виконання таких операцій:

1) на основі вхідних даних, що характеризують дану здатність корекції коду та її потужність, вибрати сімейство ІКВ з відповідними параметрами, які могли б відповідати заданим вимогам;
2) будується набір кодових комбінацій для обраного сімейства ІКВ з використанням можливостей кожного варіанту ІКВ, зокрема зворотного коду;
3) виконати попарну перевірку на множині кодових комбінацій їх кодових відстаней;
4) із множини побудованих кодових комбінацій вибирається підмножина комбінацій з кодовими відстанями, що забезпечують задану можливість корекції коду;

5) якщо побудований код відповідає заданим вимогам щодо корегувальної здатності та потужності, то обчислення завершені.

Якщо ці вимоги не можуть бути виконані, обраний набір кодових комбінацій доповнюється модифікованими числовими послідовностями, складеними з відповідних ІКВ або їх фрагментів, або вибирається сім варіантів ІКВ з більшою потужністю, після чого виконуються операції згідно з п. 2 Обчислення тривають до отримання бажаного результату.

Слід зазначити, що зі збільшенням кількості бітів багатопозиційного ІКВ-коду зростає можливість збільшення потужності коду за рахунок швидкого зростання потужності цілих сімейств ІКВ [5].

Отже, зі збільшенням розрядності багатопозиційних ІКВ-кодів розширюються їх практичні можливості не тільки в плані збільшення корекційної здатності, а й у плані збільшення потужності, що підтверджує перспективність створення новітніх інформаційних технологій на основі багатопозиційності. коди.

Висновок до розділу 3

Описані методи побудови багатопозиційних циклічних кодів з високими можливостями ідентифікації та виправлення множинних помилок базуються на використанні комбінаторних конфігурацій з кільцевою структурою, таких як ідеальні кільцеві в'язанки.

Методи дозволяють створювати системи кодування з підвищеними якісними показниками по потужності та завадостійкості.

Коригувальна ефективність циклічних кодів, побудованих на основі багатоелементних ІКВ з оптимізованими параметрами, досягає 0,5 для будь-якої великої кількості бітів кодових послідовностей.

У цьому випадку максимально можлива кількість виправних помилок становить 25% від загальної кількості розрядів коду.

Порівняно з існуючими методами синтезу корекційного коду запропоновані методи не вимагають обов’язкового пошуку генерації поліномів та побудови мінімальних незвідних поліномів у полі ГФ Галуа, як це

передбачено запропонованим алгоритмом синтезу коду Боузом, Чоудхурі та Хоквінхемом (кодів БЧХ).

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Алгоритм роботи програми

Блок-схема алгоритму програми представлена на рис. 4.1.

На початку роботи завантажуємо в програму зображення, далі введена інформація (наприклад, зображення) перетворюється в двовимірний масив. Потім генеруються зразки таблиць для перетворення. Кількість бітів до послідовність ІКВ буде за формулою (4.1).

$$e\_bit = Trunc(\log_2(S_n)) \tag{4.1}$$

В залежності від заданих початкових даних може бути e_bit = 5;

У циклі двійкові дані графічного зображення послідовно перетворюються на двійковий рядок завадостійкого циклічного коду.

Далі генеруємо вибірку ІКВ, тобто випадково робимо шум (помилки) у цій вибірці ІКВ. Потім виконується декодування і виправлення помилок.

Ідея алгоритму полягає в тому, що у мене є 2 зразки, перший - це шум, а другий - виправлений. Програма крок за кроком порівнює кожен блок зразків і підраховує кількість невідповідностей між двома зразками. Це дозволяє нам показати, скільки помилок здатний виправити алгоритм. При кількості помилок $Sum>t_2$ згідно алгоритму йде перевірка наступної послідовності вибірки ІКВ.

Наступний крок, здійснення перетворення ІКВ послідовності у масив байтів. В кінцевому результаті виводимо отримане коректуюче зображення.

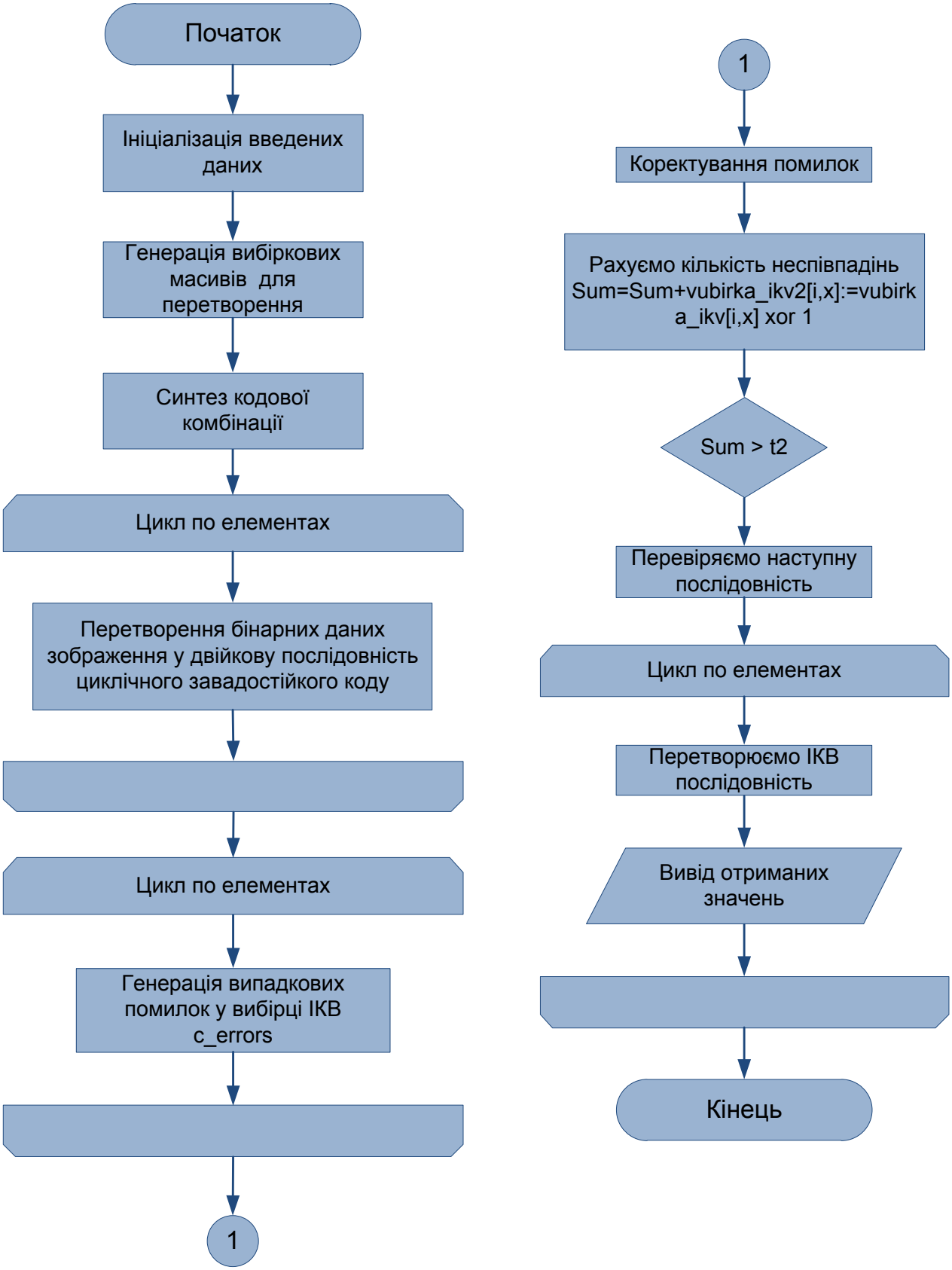

Рис. 4.1. Блок-схема алгоритму роботи програми

4.2. Дослідження результатів

Після запуску програми перед користувачем з'являється вікно, зображене на рис. 4.2. Програма встановлює послідовність ІКВ і відкриває зображення, що міститься на панелі під назвою вхідна інформація, як показано на рис. 4.2.

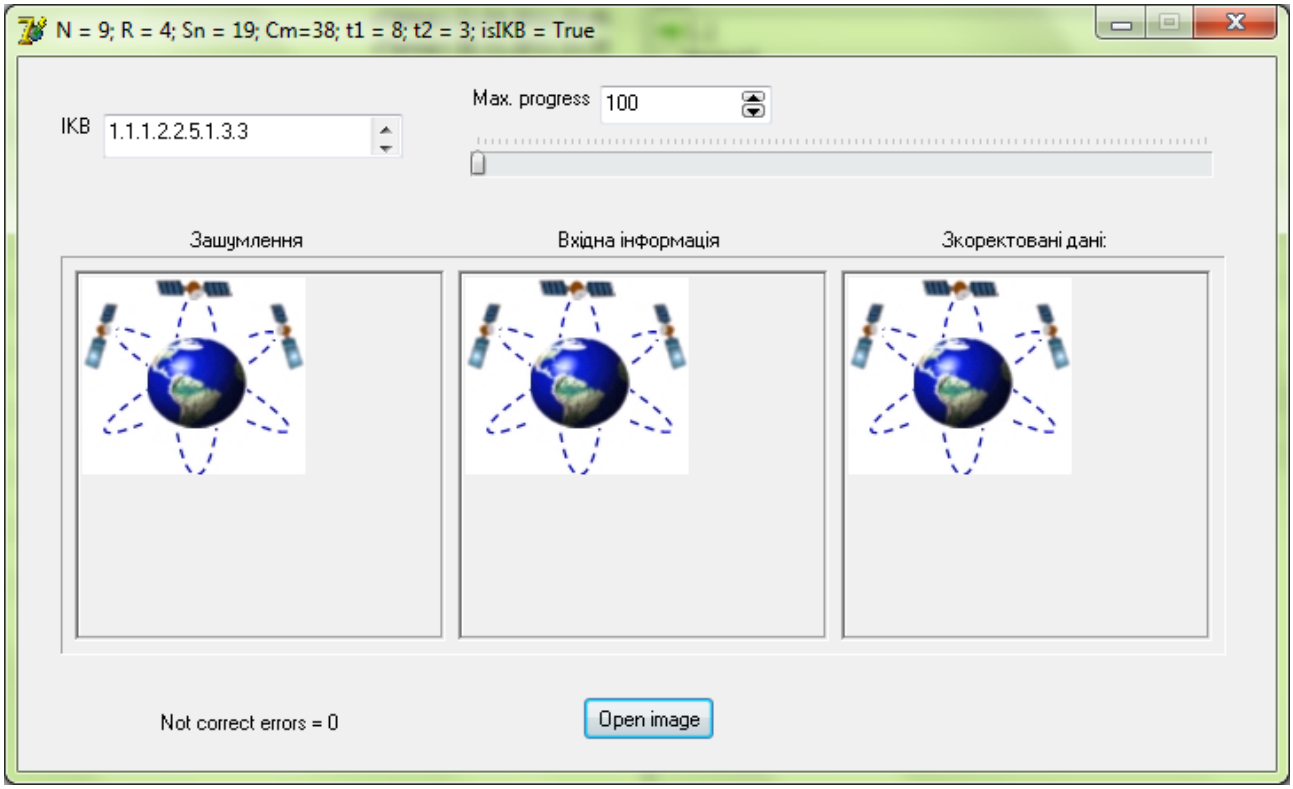

Рис. 4.2. Початок роботи програми

Наступним кроком є синтез зразка ІКВ, тобто я випадковим чином генерую шум (помилки) у зразку ІКВ, як показано на рис. 4.2. Ми можемо побачити, як на рис. 4.3. на панелі під назвою Шум. Кількість помилок можна змінити. Потім виконується декодування і виправлення помилок.

Програма крок за кроком порівнює кожен блок зразка та підраховує кількість невідповідностей. Це дозволяє нам показати, скільки помилок здатний виправити алгоритм. Якщо кількість помилок $Sum>t_2$, тоді згідно алгоритму перевіряється наступна послідовність вибірок.

Наступний крок, здійснення перетворення ІКВ послідовності у масив байтів. В кінцевому результаті виводимо отримане коректуюче зображення, де його можемо спостерігати на рис. 4.3.

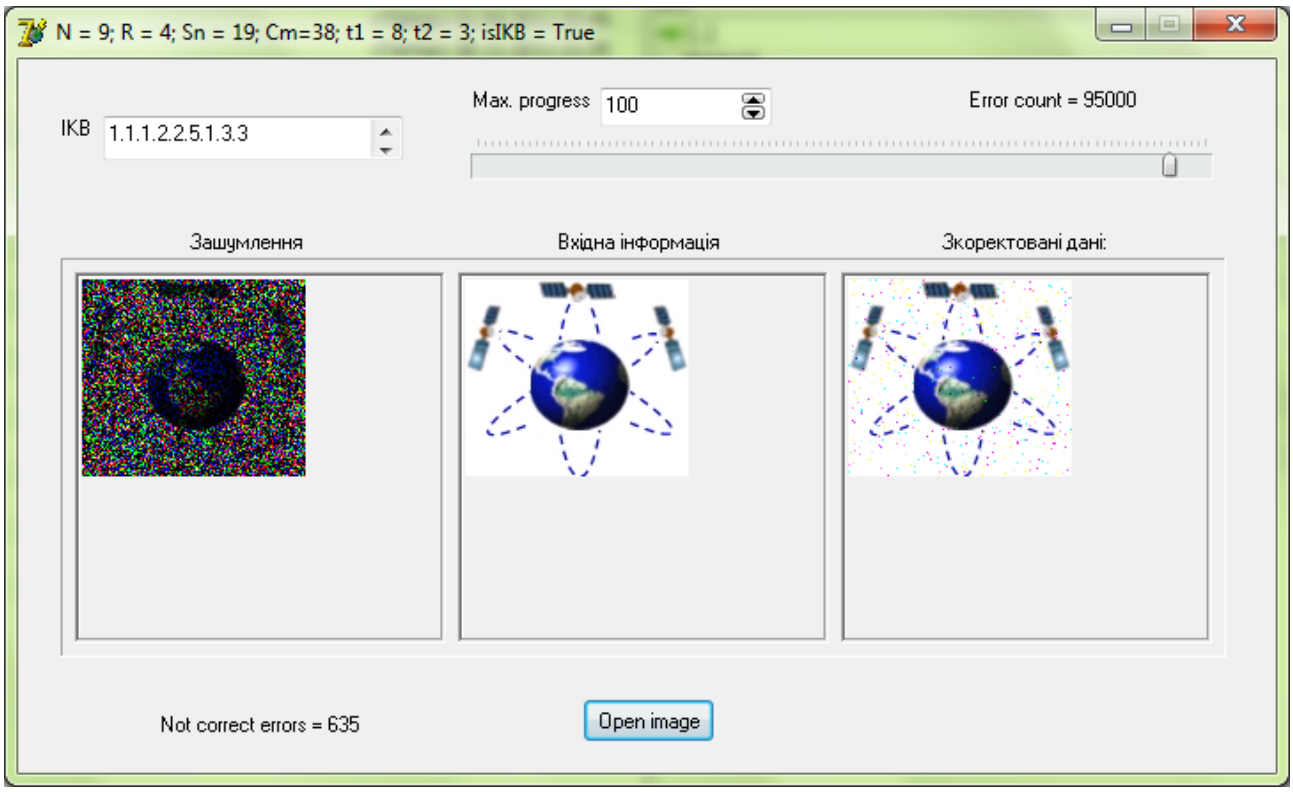

Рис. 4.3. Програмна реалізація

Аналізуючи отримані результати за прикладом, наведеним на рис. 4.3 бачимо, що коли було відкрито образ, випадковим чином створено 95 000 помилок. Програма декодування не може виправити 635 помилок.

Висновок до розділу 4

Можна зробити висновок, що програма працює дуже добре і справляється із завданням корекції та оптимізації циклічних кодів під час декодування зображення.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Основною проблемою сучасної теорії і техніки радіозв'язку і управління є підвищення стійкості телекомунікаційних систем, особливо ліній управління, як до різного роду перешкод, створюваних противником. Однією з основних концепцій стійкості, розроблених у цьому проекті, є швидке перезбирання запущеного коду, що дозволяє підвищити стійкість, енергію та параметричну секретність системи зв’язку, а також захистити інформацію від несанкціонованого доступу.

Вибрана проблема та концепція підвищення стійкості сучасних телекомунікаційних систем показують, що завдання синтезу повних класів завадостійких лінійних та нелінійних шумоподібних кодових систем є дуже актуальним. Слід також зазначити, що ефективне вирішення проблеми стійкості може бути досягнуто шляхом розробки нових інформаційних технологій на основі спільного використання перешкодостійких кодових технологій. Ці технології також мають підвищити рівень захисту інформації від несанкціонованого доступу.

5.2. Розроблення ринкової стратегії

Метою проекту було створення стійкого алгоритму кодування числових потоків даних на основі числових в’язанок і подальшої реалізації програмного забезпечення на одній з мов високого рівня.

Під час їх передачі можливий несанкціонований доступ до інформації. Щоб ускладнити використання отриманої інформації, перед передачею її можна закодувати різними кодами.

Слід зазначити, що запропонований спосіб кодування має низку переваг перед іншими кодами. Однією з них є простий алгоритм виявлення та виправлення помилок на приймальній стороні, так як поява хоча б одного символу «1» серед нулів або символу «0» серед одиниць у прийнятій кодовій комбінації свідчить про помилку. Помилка не виявляється тільки при появі

помилкового сигналу в першому або останньому символі блоку (на межі блоків нулів і одиниць). Коли в коді з’являються недійсні символи, вони негайно виявляються повністю або частково, забезпечуючи високу надійність коду. Використання цієї програми продемонструвало її високу стійкість до навмисного злому інформації та може успішно використовуватися для захисту будь-якої інформації.

5.3. Розроблення маркетингової програми

Споживачами запропонованого програмного забезпечення можуть стати користувачі Інтернету, які потребують безпечної та надійної передачі інформації.

Конкурентом продукту є SunSystem, яка випускає еквівалент цієї програми Stego.

Основними недоліками аналога є:

- обмежений обсяг передачі;
- висока вартість програмного забезпечення;
- немає вихідного коду.

Основними перевагами аналогічного рішення у порівнянні з проектним рішенням є:

- високошвидкісний код;
- можливість надсилати інформацію у кількох файлах.

5.4. Вимоги до технічного та програмного забезпечення

Програма демонструє та реалізує можливість спрощеної побудови з використанням ідеальних числових в’язанок завадостійких моделей прямого та зворотного кодів.

Досліджені та проведенні програмні експерименти для виявлення та виправлення помилок за допомогою стійких до помилок кодів на основі ідеальних числових асоціацій.

Використання завадостійкого коду, стійкого до прямих і зворотних перешкод, дозволяє підвищити безпеку при прийомі і передачі інформації.

Висновки до розділу 5

У цьому проекті з метою підвищення стійкості до відмов сучасних ІТ-систем розроблено регулярний і конструктивний метод синтезу повних класів завадостійких кодів, що дозволило істотно підвищити параметричні параметри конфіденційності та захисту інформації від несанкціонованого доступу.

ВИСНОВКИ

У магістерській роботі вирішено актуальну задачу оптимізованих циклічних кодів з використанням ідеальних кільцевих в’язанок. Були досягнуті такі результати:

1. На основі аналізу літературних джерел розглянуто методи визначення та вирішення задачі синтезу оптимізованого циклічного коду.
2. Розглянуто основні припущення методу побудови багатопозиційного завадостійкого коду на основі ідеальних кільцевих в’язанок.
3. На основі синтезу та дослідження оптимізованих циклічних кодів розроблено програмну реалізацію, що дозволяє коригувати та виправляти помилки під час декодування.
4. Розроблено програму візуального відображення для візуального представлення результатів декодування.
5. Проведено дослідження впливу кількості допущених помилок на кінцевий результат програми.
6. Реалізація програмного забезпечення була протестована для максимально ефективного виправлення помилок, щоб зображення можна було розпізнати після декодування.

Отримані результати та впровадження програмного забезпечення можуть стати основою для достовірності інформації, підвищення надійності, функціональної безпеки та життєздатності інформації та систем управління інформацією.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Хохлов Г. И.* Злементьі теории корректирующих кодов / МИРЗА. — М., 1980. — 136 с.
2. Цымбал В.П. Теория информации и кодирование. – К.:Вища школа, 1982. – 304 с.
3. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки. – М.: Сов. Радио, 1974. – 590 с.
4. Берлэкэмп Э.Алгебраическая теория кодирования. – М.: Мир, 1971. – 478 с.
5. Різник В.В. Синтез оптимальних комбінаторних систем. – Львів: Вища школа, 1989. – 168 с.
6. Бандирська О. В. Досконалі системи мір як свідчення предвічної гармонії Всесвіту // Світоглядні читання до 200-річчя Ч. Дарвіна: зб. наук. праць. – К.: Четверта хвиля, 2010. – С.49–53.
7. Riznyk V. Applicationof the Gold Ring Bundles for innovative non-redundant radar or sonar systems // European Physical
8. Journal (EPJ-SP), v.154, Febryary, 2008. – Pp.183–186.
9. Жураковський Ю. П., Полторак В. П. Ж91 Теорія інформації та кодування: Підручник. — К.: Вища шк., 2001. — 255 с.: іл.
10. *Стратонович Р. Л.* Теория информации. — М.: Сов. радио, 1975. — 420 с.
11. *Горяиное О. А., Хохлов Г. И.* Злементм теории информации и кодирова- ния / МИРЗА. — М., 1985. — 117 с.
12. *Блейхут Р.* Теория и практика кодов, контролирующих ошибки. — М.: Мир, 1986. — 576 с.
13. http://uk.wikipedia.org/wiki/Ідеальна_кільцева_в’язанка
14. http://uk.wikipedia.org/wiki/Коди_БЧХ
15. http://uk.wikipedia.org/wiki/властивоті_кодів
16. http://uk.wikipedia.org/wiki/потужність_коду
17. http://uk.wikipedia.org/wiki/завадостійке_кодування
18. http://uk.wikipedia.org/wiki/Інверсний_код
19. http://uk.wikipedia.org/wiki/циклічні_коди

20. http://uk.wikipedia.org/wiki/Ideal_Ring_Bundles

ДОДАТОК А. КОД ПРОГРАМИ КОДУВАННЯ

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, ExtCtrls,math,matem, XPMan, Gauges,RichEdit,jpeg,
  ExtDlgs, Spin;

type
  int = integer;
  str = string;
  pmas_2i=^mas_2i;
  mas_2i = array of mas_i;
  Tmas_v= array of str;

  TForm1 = class(TForm)
    Label3: TLabel;
    od: TOpenDialog;
    Label2: TLabel;
    Label1: TLabel;
    mmo1: TMemo;
    lbl1: TLabel;
    xpmnfst1: TXPManifest;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    PaintBox1: TPaintBox;
    PaintBox2: TPaintBox;
    PaintBox3: TPaintBox;
    btn2: TButton;
    OpenPictureDialog1: TOpenPictureDialog;
    Label4: TLabel;
    Label5: TLabel;
    trckbr1: TTrackBar;
    Label6: TLabel;
    se1: TSpinEdit;
    lbl2: TLabel;
    lbl3: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure btn1Click(Sender: TObject);
    procedure btn2Click(Sender: TObject);
    procedure trckbr1Change(Sender: TObject);
    procedure se1Change(Sender: TObject);  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  c_bit:int;
  error:int;
  sn,count:int;
  _in,_er,_out:TStringList;
  c_lines:int;
  scroll_n:int;
  mmax,max_bit:int;
```

```
  generate:bool;
  Pos_visible_line:int;
  bitmap:TBitmap;
  b_org,b_err,b_ecc:TBitmap;
  a:TIkb_param;
  c_rec_ikv:int;
  errors:int;
  matrix2:mas_2i;
  vubirka:mas_i;
  vubirka2:mas_i;
  vubirka_org:mas_i;
  vubirka_ikv:mas_2i;
  vubirka_ikv2:mas_2i;


  implementation

{$R *.dfm}

function find_d (s:mas_i):int;
var v:str;
max,max2,max_id,i,sum,sum2,j:int;
begin
max:=MaxInt;
max2:=a.n - a.R -1;
max_id:=0;
for i:=0 to sn*2-1 do
  begin
   sum:=0;
   for j:=0 to sn-1 do
     begin
        SUM:=  SUM +((matrix2[i,j] xor s[j]));
        if sum > max2 then Break;
     end;
    if (max > sum)  then
    begin
      max:=sum;
      max_id:=i;
    end;
  end;
//Result:=matrix2[max_id];
Result:=max_id;
end;

procedure i2b(x:Byte;p:pint;count:int=8);
var i,k:byte;
begin
i:= 1 shl (count-1);
k:=0;
while i>0 do
begin
  p^:=sign(x and i);
  Inc(p);
  i:= i shr 1;
end;

end;


procedure b2i2(p:pint;var x:byte;count:int=8);
var i,k:byte;
begin
i:= 1 shl (count-1);
```

```
x:=0;
for k:=0 to count-1 do
  begin
  if p^=1 then
  x:=x or i;
  i:=i shr 1;
  Inc(p);
  end;

end;

{procedure i2b(x:byte; p:pmas_i;count:int) ; assembler;
asm
  pushad
  mov bl,x
  mov edi,p
  push ecx
  mov esi,8
  sub esi,count
  mov ecx,esi
  rcl bl,cl
  pop ecx
  xor eax,eax
@m1:
    and al,0
    rcl bl,1
    adc al,0
    stosd
    loop @m1
popad
end;

procedure b2i2(p:pmas_i;var x:byte;count:int); assembler;
asm
  pushad
  mov esi,p
  mov edi,x
  mov ecx,count
  xor edx,edx
@m1:
    lodsd
    rcr eax,1
    rcl edx,1
loop @m1
    mov [edi],dl
popad
end;
}

procedure timer;
function decode(x:mas_i):int ;
var i,j,c_error:int;
    r:BOOL;
begin
Result:=0;
for i:=0 to High(matrix2)do
 begin
    r:=True;
    for j:=0 to Sn-1 do
      if x[j] <> matrix2[i][j] then
      begin
        r:=False;
        break;
```

```
      end;
    if r then
    begin
      Result:=i;
      Exit;
    end;
  end;

end;

var x,y,i,c_error:int;
    b:Byte;
    p,p1:PByteArray;
begin
  with Form1 do
  begin
     FillMemory(@vubirka2[0],c_rec_ikv*19,0);
     c_error:=0;
     for y:=0 to c_rec_ikv-2 do
      for i:=0 to a.sn-1 do
      begin
        vubirka_ikv2[y,i]:=vubirka_ikv[y,i];
      end;
     for y:=0 to errors do
      begin
       x:=random(sn);
       i:=Random(c_rec_ikv-2);
       vubirka_ikv2[i,x]:=vubirka_ikv[i,x] xor 1; // create errors
      end;
  i:=0;
  for y:=0 to c_rec_ikv-2 do
    begin
      b:=find_d(vubirka_ikv2[y]);
      i2b(b,@vubirka2[i],c_bit);
      if not CompareMem(@vubirka_org[i],@vubirka2[i],20) then Inc(c_error);
      b:=decode(vubirka_ikv2[y]);
      i2b(b,@vubirka[i],c_bit);

      inc(i,c_bit);

    end;
  i:=0;
  for y:=0 to b_err.Height-1 do
    begin
      p:=b_err.ScanLine[y];
      p1:=b_ecc.ScanLine[y];
      for x:=0 to b_err.Width*3-1 do
        begin
          b2i2(@vubirka[i],p[x]);
          b2i2(@vubirka2[i],p1[x]);
          Inc(i,8);
        end;
    end;


  if Assigned(b_org) then

BitBlt(PaintBox2.Canvas.Handle,0,0,PaintBox2.ClientWidth,PaintBox2.ClientHeight,b_org.Canvas.Handle,0,scroll_n,
SRCCOPY);
  if Assigned(b_err) then

BitBlt(PaintBox1.Canvas.Handle,0,0,PaintBox1.ClientWidth,PaintBox1.ClientHeight,b_err.Canvas.Handle,0,scroll_n,S
RCCOPY);
```

```
  if Assigned(b_ecc) then

BitBlt(PaintBox3.Canvas.Handle,0,0,PaintBox2.ClientWidth,PaintBox2.ClientHeight,b_ecc.Canvas.Handle,0,scroll_n,
SRCCOPY);
  lbl3.Caption:='Not correct errors = '+inttostr(c_error);
  end;
end;

function StrToIKB(s:str; var m:mas_i):bool;
var p,pbeg:pchar;
    len:int;
begin
s[Pos(#13,s)]:=#0;
s:=pchar(s);
s:=s+','#0;
len:=0;
p:=@s[2];
pbeg:=p-1;
while p^<>#0 do
  begin
    if p^in [',','.'] then
      begin
        p^:=#0;
        inc(len);
        SetLength(m,len);
        m[len-1]:=strtoint(pbeg);
        inc(p);
        pbeg:=p;
      end else inc(p);
  end;
Result:=true;
end;

function rolmas(m:mas_i; i:int; len:int):mas_i;
var slen:int;
    j:int;
begin
  slen:=len;//len;
  j:=0;
  SetLength(Result,slen);
  while bool(slen)do
    begin
      Result[j]:=m[i];
      inc(i);
      inc(j);
      if i>= len then i:=0;
      dec(slen);
    end;
end;

function IKBToBarker(m:mas_i):mas_i;
var len:int;
    i,count,j:int;
    p1:pint;
begin
len:=0;
for i:=0 to high(m) do
  len:=len+m[i];
SetLength(Result,len);
    p1:=@Result[0];
    count:=high(m)+1;
    p1^:=1;
    inc(p1);
```

```
    for j:=0 to count-2 do
      begin
        FillMemory(p1,(m[j]-1)*4,0);
        Inc(p1,m[j]-1);
        p1^:=1;
        inc(p1);
      end;
    FillMemory(p1,(m[j]-1)*4,0);
end;

function Bild_Matrix(m:mas_i):Tmas_v;
var len,len2:int;
    i,j:int;
    v:mas_i;
begin
  len:=high(m)+1;
  count:=len*2;
  len2:=mmax;//len;
  SetLength(Result,len*2);
  for i:= 0 to len-1 do
    begin
     v:=rolmas(m,i,len);
     for j:=1 to len2 do
      begin
        Result[i]:=Result[i]+inttostr(v[j-1]);
        if v[j-1] = 0 then
        Result[len+i]:=Result[len+i]+'1' else
        Result[len+i]:=Result[len+i]+'0';
      end;
    end;
end;

function Bild_Matrix2(m:mas_i):mas_2i;
var len,len2:int;
    i,j:int;
    v:mas_i;
begin
  len:=high(m)+1;
  count:=len*2;
  len2:=len;//len;
  SetLength(Result,len*2);
  for i:= 0 to len-1 do
    begin
     SetLength(Result[i],len);
     SetLength(Result[len+i],len);
     v:=rolmas(m,i,len);
     for j:=1 to len2 do
      begin
        Result[i,j-1]:=v[j-1];
        if v[j-1] = 0 then
        Result[len+i,j-1]:=1 else
        Result[len+i,j-1]:=0;
      end;
    end;
end;

function rorstr(s:str; i:int):str;
var
  c:char;
  len,j,slen:int;
begin
  len:=length(s);
  slen:=len;
```

```
 j:=1;
 SetLength(Result,len);
 while bool(slen) do
   begin
     Result[j]:=s[i];
     inc(i);
     inc(j);
     if i>= len then i:=1;
     dec(slen);
   end;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
od.InitialDir:=GetCurrentDir+'\';
Randomize();
SetTimer(0,12,10,@timer)
end;

procedure mas2bin(m:mas_b);
var i,j,ma,ma2:int;
    tmp:str;
    p1:PByte;
    p2:pchar;
    d:Byte;
function b2i(p:pchar;d:int):int ;
var i,j,ma,ma2:int;
    tmp:str;
    p1:PByte;
    p2:pchar;
begin
  i:=1 shl (d-1);
  Result:=0;
  for j:=1 to d do
   begin
     if p^='1' then
       Result:=Result or i;
     Inc(p);
     i:=i shr 1;
   end;
end;

begin
 ma:=High(m)+1;
 i:=ma mod max_bit;
 ma2:=ma*8+i;
 SetLength(tmp,ma2);
 FillMemory(@tmp[1],ma2,48);
 p1:=@m[0];
 p2:=@tmp[1];
 for i:=1 to ma do
   begin
     d:=1 shl 7;
     for j:=0 to 7 do
       begin
         if p1^ and d <>0 then
           p2^:='1';
         Inc(p2);
         d:= d shr 1;
       end;
     Inc(p1);
   end;
```

```
  i:=ma2 div max_bit;
  SetLength(m,i);
  p2:=@tmp[1];
  p1:=@m[0];
  while i>0 do
  begin
    p1^:=b2i(p2,max_bit);
    Inc(p2,max_bit);
    dec(i);
  end;
end;

//1,1,1,2,2,5,1,3,3
procedure TForm1.Button1Click(Sender: TObject);
var
    s1:str;
    m:mas_i;
    s:mas_b;
    k,k1,p,a:str;
    i,j,r,q:int;
    ID:DWORD;
    x,y:int;
begin

{ if Assigned(_in) then
  _in.Free;
_in:=TStringList.Create;
if Assigned(_er) then
  _er.Free;
_er:=TStringList.Create;
if Assigned(_out) then
  _out.Free;
_out:=TStringList.Create;

StrToIKB(mmo1.Text,m);
mmax:=strtoint(edit2.Text);
m:=IKBToBarker(m);
Revers(m);
matrix:=Bild_Matrix(m);
sn:=length(matrix[0]);
error:=StrToInt(LabeledEdit1.Text);
SetLength(s1,1000);
FillMemory(PChar(s1),1000,ord(' '));
x:=0;y:=0;
for i:=0 to high(vubirka)do
   begin
     k:=matrix[vubirka[i]];
     _in.Add(k+s1);
     for j:=1 to error do
       begin
         r:=random(sn-1)+1;
         if k[r]='0' then
           k[r]:='1' else
           k[r]:='0';
       end;
       _er.Add(k+s1);


       k1:=find_d(k);
       _out.Add(K1+s1);
       if k1 <> matrix[vubirka[i]] then
         bitmap.Canvas.Rectangle(x,y,x+5,y+10);
       Inc(x,5);
```

```
      if x >= 880 then
       begin
         x:=0;
         Inc(y,10);
       end;
     end;
if Assigned(bitmap) then
BitBlt(PaintBox1.Canvas.Handle,0,0,PaintBox1.ClientWidth,PaintBox1.ClientHeight,bitmap.Canvas.Handle,0,scroll_n
,SRCCOPY);
id:=0;
  for i:=0 to _in.Count-1 do
    begin
      k:=_out.Strings[i];
      p:=_in.Strings[i];
      if (k<>'')and (p<>'')and(k<>p)then
      for j:=1 to Length(k) do
      begin
        if (k=' ')and(p=' ')then break;
        if k[j]<>p[j] then
           inc(id);
      end;
    end;
mmohistory.Lines.Add(Format('[%s]: %s [ %d ];'#13#10,[timeToStr(now),'Кількість не визначених помилок',id]));
}
end;


procedure TForm1.btn1Click(Sender: TObject);
var N:int;
    m:mas_i;
    t1,t2:int;
    s:str;
begin
  StrToIKB(mmo1.Text,m);
  a:=Get_RN(m);
  if (a.Sn >= 4*(a.N-a.R))then
    begin
      s:='Sn >= 4(n-R)';
      t1:=2*(a.N-a.R)-1;
      t2:=a.N-a.R-1;
    end
  else
    begin
      s:='Sn < 4(n-R)';
      t1:=a.Sn-2*(a.N-a.R)-1;
      t2:=(a.Sn-2*(a.N-a.R+1))shr 1;
    end;
  generate:=False;
  Form1.Caption:=Format('N = %d; R = %d; Sn = %d; t1 = %d; t2 = %d; isIKB =
%s',[a.N,a.R,a.Sn,t1,t2,BoolToStr(is_ikb(m),true)]);
end;

procedure TForm1.btn2Click(Sender: TObject);
var max_chuslo:int;
    y,x,i:int;
    poss:int;
    j:TJPEGImage;
    p:PByteArray;

    b:Byte;
    m:mas_i;
var N:int;
    t1,t2:int;
```

```
    s:str;
begin
  StrToIKB(mmo1.Text,m);
  a:=Get_RN(m);
  if (a.Sn >= 4*(a.N-a.R))then
    begin
      s:='Sn >= 4(n-R)';
      t1:=2*(a.N-a.R)-1;
      t2:=a.N-a.R-1;
    end
  else
    begin
      s:='Sn < 4(n-R)';
      t1:=a.Sn-2*(a.N-a.R)-1;
      t2:=(a.Sn-2*(a.N-a.R+1))shr 1;
    end;
  generate:=False;
  Form1.Caption:=Format('N = %d; R = %d; Sn = %d; Cm=%d; t1 = %d; t2 = %d; isIKB =
%s',[a.N,a.R,a.Sn,a.Sn*2,t1,t2,BoolToStr(is_ikb(m),true)]);

  OpenPictureDialog1.InitialDir:=GetCurrentDir;
if  OpenPictureDialog1.Execute  and (OpenPictureDialog1.FileName <> '') then
  begin
    if Assigned(b_org) then
      b_org.Free;
    if Assigned(b_err) then
      b_err.Free;
    if Assigned(b_ecc) then
      b_ecc.Free;
    b_org:=TBitmap.Create;
    b_err:=TBitmap.Create;
    b_ecc:=TBitmap.Create;
    if LowerCase(ExtractFileExt(OpenPictureDialog1.FileName)) = '.jpg' then
      begin
        j:= TJPEGImage.Create;
        j.LoadFromFile(OpenPictureDialog1.FileName);
        b_org.Assign(j);
        j.Free;
      end
      else
        b_org.LoadFromFile(OpenPictureDialog1.FileName);
    b_err.Width:=b_org.Width;
    b_err.Height:=b_org.Height;
    b_err.PixelFormat:=pf24bit;
    b_org.PixelFormat:=pf24bit;
    b_ecc.Width:=b_org.Width;
    b_ecc.Height:=b_org.Height;
    b_ecc.PixelFormat:=pf24bit;
    SetLength(vubirka,b_org.Width*b_org.Height*3*8);
    SetLength(vubirka2,b_org.Width*b_org.Height*3*8);
    SetLength(vubirka_org,b_org.Width*b_org.Height*3*8);

    i:=0;
    for y:=0 to b_org.Height-1 do
      begin
        p:=b_org.ScanLine[y];
        for x:=0 to b_org.Width*3-1do
          begin
            i2b(p[x],@vubirka[i]);
            i2b(p[x],@vubirka_org[i]);
            Inc(i,8);
          end;
      end;
```

```
    generate:=True;

    StrToIKB(mmo1.Text,m);
    m:=IKBToBarker(m);
    matrix2:=Bild_Matrix2(m);
    sn:=a.Sn;
    c_bit:=Trunc(Log2(a.Sn*2-1));
    c_rec_ikv:=((High(vubirka)+1) div c_bit+1);
    SetLength(vubirka_ikv,c_rec_ikv);
    SetLength(vubirka_ikv2,c_rec_ikv);
    i:=0;y:=0;
    while i<High(vubirka) do
    begin
     b2i2(@vubirka[i],b,c_bit);
     Inc(i,c_bit);
     SetLength(vubirka_ikv[y],A.Sn);
     SetLength(vubirka_ikv2[y],A.Sn);
     for x:=0 to a.Sn-1 do
        vubirka_ikv[y,x]:=matrix2[b,x];
     Inc(y);
    end;
  end;
  FillMemory(@vubirka[0],High(vubirka)*4,0);
end;

procedure TForm1.trckbr1Change(Sender: TObject);
begin
 errors:=trckbr1.Position * 1000;
 lbl2.Caption:='Error count = '+inttostr(errors);
end;

procedure TForm1.se1Change(Sender: TObject);
begin
trckbr1.Max:=se1.Value;

end;

end.
```

ДОДАТОК Б. КОД ПРОГРАМИ ДЕКОДУВАННЯ

```
unit matem;
interface
uses windows,SysUtils;
type
  int = integer;
  str = string;
  Pmas_i = ^mas_i;
  Pmas_b = ^mas_b;
  mas_i=array of int;
  Matrix_mas_i = array of mas_i;

  mas_b=array of byte;
  mas_s= array of str;
  TIkb_param = record
                 N,R,Sn:int;
               end;

function is_ikb(barker:mas_b):boolean;overload;
function is_ikb(ikb:mas_i):boolean;overload;
function is_ikb(s:str):boolean;overload;
function StrToMas(s:str; var m:mas_i):boolean;
function rolmas(m:mas_b; i:int; len:int):mas_b;
function IKBToBarker(m:mas_i):mas_b;
function BarkertoIKB(m:mas_b):mas_i;
function bild_Suprov_matrix(a:mas_i;Sn,p:int):mas_i;
function bild_rinz(ikv:mas_i;Sn:int):Matrix_mas_i;
function Get_RN(m:mas_i):TIkb_param;overload;
function Get_RN(m:mas_b):TIkb_param;overload;

implementation
uses Math;
var ikb_p:TIkb_param;

function C_1(m1,m2:Pmas_b;len:int):int;assembler;
asm
  pushad
  mov edi,m1
  mov esi,m2
  mov ecx,len
  xor edx,edx
  xor eax,eax
@1:
  mov al,[edi]
  mov ah,[esi]
  and al,ah
  xor ah,ah
  add edx,eax
  inc esi
  inc edi
loop @1
  mov Result,edx
  popad
end;

function rolmas(m:mas_b; i:int; len:int):mas_b;
var slen:int;
    j:int;
begin
  slen:=len;
```

```
 j:=0;
 SetLength(Result,slen);
 while boolean(slen)do
   begin
     Result[j]:=m[i];
     inc(i);
     inc(j);
     if i>= len then i:=0;
     dec(slen);
   end;
end;

function Get_RN(m:mas_i):TIkb_param;overload;
var i:int;
begin
 Result.R:=0;
 Result.N:=high(m)+1;
 Result.Sn:=SumInt(m);
 for i:=0 to high(m)do
     if m[i]=1 then
       inc(Result.R);
end;

function SumByte(m:mas_b):int;
var i:int;
begin
 Result:=0;
 for i:=0 to High(m)do
    Result:=Result+m[i];
end;

function Get_RN(m:mas_b):TIkb_param;overload;
var i:int;
   m0:mas_b;
begin
 Result.N:=High(m)+1-SumByte(m);
 m0:=rolmas(m,1,high(m)+1);
 Result.R:=C_1(@m[0],@m0[0],high(m)+1);
 Result.Sn:=high(m)+1;
end;

function StrToMas(s:str; var m:mas_i):boolean;
var p,pbeg:pchar;
   len:int;
begin
s:=s+','#0;
for len:=1 to length(s) do
 if s[len] in [' ','.','-'] then
    s[len]:=',';
len:=0;
p:=@s[2];
pbeg:=p-1;
while p^<>#0 do
 begin
   if p^=',' then
     begin
       p^:=#0;
       inc(len);
       SetLength(m,len);
       m[len-1]:=strtoint(pbeg);
       inc(p);
       pbeg:=p;
     end else inc(p);
```

```
  end;
Result:=true;
end;

function IKBToBarker(m:mas_i):mas_b;
var len:int;
    i,count,j:int;
    p1:pbyte;
begin
len:=0;
for i:=0 to high(m) do
  len:=len+m[i];
SetLength(Result,len);
    p1:=@Result[0];
    count:=high(m)+1;
    p1^:=1;
    inc(p1);
    for j:=0 to count-2 do
      begin
        FillMemory(p1,(m[j]-1),0);
        Inc(p1,m[j]-1);
        p1^:=1;
        inc(p1);
      end;
    FillMemory(p1,(m[j]-1),0);
end;

function is_ikb(s:str):boolean;overload;
var ikb:mas_i;
begin
StrToMas(s,ikb);
Result:=is_ikb(ikb);
end;

function is_ikb(ikb:mas_i):boolean;
var
    barker:mas_b;
begin
ikb_p:=Get_RN(ikb);
barker:=IKBToBarker(ikb);
result:=is_ikb(barker);
end;

function is_ikb(barker:mas_b):boolean;overload;
var i:int;
    count:int;
    barker2:mas_b;
begin
Result:=true;
ikb_p:=Get_RN(barker);
count:=ikb_p.Sn;
 for i:=1 to count-1 do
  begin
    barker2:=rolmas(barker,i,count);
    if ikb_p.R<>C_1(@Barker[0],@Barker2[0],count) then
      begin
        Result:=false;
        exit;
      end;
  end;
end;

function BarkertoIKB(m:mas_b):mas_i;
```

```
var i:int;
    c:int;
    c_r:int;
begin
c:=1;
c_r:=0;
SetLength(m,high(m)+2);
m[high(m)]:=1;
for i:=1 to high(m)do
    if m[i]=1 then
      begin
       inc(c_r);
       SetLength(Result,c_r);
       Result[c_r-1]:=c;
       c:=1;
      end else inc(c);
end;

function bild_Suprov_matrix(a:mas_i;Sn,p:int):mas_i;
var matrix:array of mas_i;
    c_a,c:int;
    i,j,z,pp:int;
    s:int;
    rez_vector:mas_i;
    rez_c:int;
begin
//bilding
rez_c:=0;
c_a:=high(a)+1;
SetLength(matrix,c_a);
SetLength(rez_vector,c_a);
pp:=-1;
for i:=0 to c_a-1 do
begin
  SetLength(matrix[i],c_a);
  for j:=0 to c_a-1 do
  begin
   if pp=j then
      matrix[i,j]:=1;
   if j=c_a-1 then
    matrix[i,j]:=a[i];
  end;
  inc(pp);
end;
// mul
a[0]:=1;
for i:=1 to c_a-1 do
  a[i]:=0;
c:=1;
for z:=1 to Sn do
  begin
   for i:=0 to c_a-1 do // N riadok
      begin
        s:=0;
        for j:=0 to c_a-1 do
            s:=s+matrix[i,j]*a[j];
        rez_vector[i]:=s mod p;
      end;
   if rez_vector[c_a-1]=0 then
    begin
    inc(rez_c);
    SetLength(Result,rez_c);
    Result[rez_c-1]:=c;
```

```
    c:=1;
    end else inc(c);
    for i:=0 to c_a-1 do
      a[i]:=rez_vector[i];
  end;
end;

procedure sort(var m:mas_i);
var i,j:int;
begin
i:=0;
while i<high(m)do
  if m[i]>m[i+1] then
    begin
      j:=m[i];
      m[i]:=m[i+1];
      m[i+1]:=j;
      i:=0;
    end else inc(i);
end;

function bild_rinz(ikv:mas_i;Sn:int):Matrix_mas_i;
  var i,j:int;
      len:int;
      r_mas,r_mas2:mas_i;
begin
SetLength(Result,Sn-1);
len:=high(ikv)+2;
SetLength(r_mas2,len);
SetLength(r_mas,len);
r_mas[len-1]:=SumInt(ikv);
r_mas2[len-1]:=r_mas[len-1];
for i:=high(ikv) downto 0 do
   r_mas[i]:=r_mas[i+1]-ikv[i];
for i:=1 to Sn-1 do
  begin
    for j:=0 to len-2 do
      r_mas2[j]:=(r_mas[j]*i)mod Sn ;
    SetLength(Result[i-1],high(ikv)+1);
    sort(r_mas2);
    for j:=high(r_mas2) downto 1 do
        Result[i-1,j-1]:=r_mas2[j]-r_mas2[j-1];
  end;
end;
end.
```