Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Кваліфікаційна наукова
праця на правах рукопису

СУЛЕМА ОЛЬГА КОСТЯНТИНІВНА

УДК 004.627

ДИСЕРТАЦІЯ

АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПРОЦЕСІВ АВТОМАТИЧНОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ ЛОГІСТИКИ НА ОСНОВІ ШТРИХОВИХ КОДІВ З ТРЬОМА ГРАДАЦІЯМИ КОЛЬОРУ

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ О.К. Сулема
(підпис)

Науковий керівник: Дичка Іван Андрійович, доктор технічних наук, професор

Київ – 2021

АНОТАЦІЯ

Сулема О. К. Алгоритмічне та програмне забезпечення процесів автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору. − Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2021.

Автоматична ідентифікація об’єктів є важливим аспектом у багатьох галузях людської діяльності, однією з яких є логістика. Застосування технологій автоматичної ідентифікації об’єктів логістики дозволяє спростити відстежування їхнього місцезнаходження під час транспортування та складського зберігання, а також забезпечити процеси контролю, звітування тощо. Це позитивно впливає на інтенсифікацію виробництва, торгівлі, зокрема міжнародної, поштових сервісів тощо завдяки безпомилковому та швидкому доступу до інформації про об’єкти обліку, а також автоматичному збору даних про них та оновлення цих даних у програмних системах без втручання людини.

Автоматична ідентифікація об’єктів може ґрунтуватись на технологіях подання даних на основі штрихових кодів, радіочастотної ідентифікації, смарт-карт тощо. Однією з найпоширеніших технологій автоматичної ідентифікації об’єктів є технологія на основі штрихового кодування інформації, яка забезпечує високу точність та швидкість введення інформації до комп’ютерних систем та є економічно привабливою завдяки низькій вартості витратних матеріалів і доступному обладнанню.

На сьогодні існує низка підходів до подання даних у вигляді штрихових кодів, проте досі залишаються актуальними задачі підвищення

щільності подання інформації у вигляді штрихових кодів та підвищення завадостійкості штрихкодових позначок.

Іншою важливою проблемою, пов’язаною зі створенням програмних систем на основі технології штрихового кодування для галузі логістики, є відсутність проблемно-орієнтованого підходу до задачі інтегрування елементів технології штрихового кодування при розробленні програмного забезпечення процесів автоматичної ідентифікації об’єктів.

Наявність зазначених актуальних задач та проблем визначає актуальну науково-технічну задачу підвищення ефективності розроблення алгоритмічного та програмного забезпечення процесів автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору, яка вирішується у цій дисертаційній роботі.

Метою дисертаційної роботи є підвищення ефективності логістичних систем за рахунок автоматизації виробничих процесів на основі високощільного штрихового кодування даних про об’єкти логістики.

У першому розділі дисертаційної роботи проаналізовано алгоритмічно-програмні рішення для автоматичної ідентифікації об’єктів у галузі логістики, зокрема вивчено сучасний стан логістичного програмного забезпечення, досліджено методи автоматичної ідентифікації об’єктів та особливості оброблення штрихкодованої інформації, що дозволило сформувати основні вимоги до програмного забезпечення процесів автоматичної ідентифікації об’єктів логістики.

У другому розділі розроблено алгоритмічне та програмне забезпечення процесів формування даних на основі штрихових кодів з трьома градаціями кольору, зокрема сформульовано задачу забезпечення компактного подання даних у вигляді штрихкодових зображень, розроблено й досліджено алгоритмічне та програмне забезпечення процесів ущільнення даних при формуванні штрихкодових позначок,

розроблено й досліджено алгоритмічне та програмне забезпечення процесів формування дворівневих штрихкодових позначок.

У третьому розділі розроблено алгоритмічне та програмне забезпечення завадостійкості штрихових кодів з трьома градаціями кольору, зокрема сформульовано задачу забезпечення завадостійкості штрихкодових зображень, розроблено й досліджено алгоритмічне та програмне забезпечення завадостійкості штрихових кодів з трьома градаціями кольору.

Четвертий розділ присвячено створенню елементів технології проєктування програмного забезпечення процесів автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, зокрема запропоновано шкалу пріоритизації вимог до розроблюваної програмної системи, сформульовано вимоги до програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, запропоновано базову архітектуру програмної системи автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору, розроблено шаблон проєктування «Перетворювач», визначено основні етапи технології проєктування програмних засобів для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору.

У дисертаційній роботі отримано низку **нових наукових результатів**, зокрема **уперше** запропоновано базову архітектуру програмної системи автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, використання якої дозволяє спростити процес розроблення програмного забезпечення систем автоматичної ідентифікації об’єктів та яка, на відміну від існуючих, створює можливість дворівневого доступу до інформації про об’єкт логістики та підвищення щільності подання даних завдяки використанню дворівневих штрихових кодів з трьома градаціями кольору.

Уперше розроблено структурний метод ущільнення алфавітно-цифрових даних, що підлягають поданню у вигляді штрихового коду з трьома градаціями кольору, який передбачає використання, крім комп'ютерного алфавіту ASCII, кількох додаткових алфавітів з потужностями, меншими за 256 (потужність ASCII), та ґрунтується на розбитті вхідної алфавітно-цифрової послідовності на суміжні підпослідовності символів та їх перетворенні у триколірні (трійкові) штрихкодові знаки, сумарна довжина яких менша, аніж трійкова довжина вхідних повідомлень, що підвищує інформаційну щільність подання даних на носії у середньому в 1,16–1,5 разів та переважає аналогічний показник у разі застосування для ущільнення статистичних методів (на основі кодів Хаффмана, Шеннона-Фано).

Уперше розроблено математичну модель, яка дозволяє здійснювати вибір оптимальної кількості додаткових (окрім ASCII) алфавітів для використання при перетворенні вхідних алфавітно-цифрових даних, що підлягають поданню у вигляді штрихових кодів з трьома градаціями кольору, та визначати оптимальні потужності цих алфавітів і тип перетворення підпослідовностей суміжних символів у числову форму, за яких досягається максимально можливе інформаційне ущільнення даних на штрихкодовому носії.

Уперше розроблено алгоритмічне забезпечення процесу завадозахищеного кодування штрихкодових позначок з трьома градаціями кольору, визначальною рисою якого є застосування обчислень у скінченому полі $GF(3^s)$ за модулем незвідного многочлена степеня s, що забезпечує виправлення спотворень двох видів – помилок і стирань, та уможливлює відновлення даних при ушкодженні до 37,5% площі штрихкодового зображення.

Уперше розроблено метод дворівневого штрихового кодування двох незалежних наборів даних та алгоритмічне забезпечення процесів формування штрихкодових позначок з трьома градаціями кольору, характерною рисою яких є застосування процедур визначення контрольного біта для подання другого набору даних та створення дворівневої штрихкодової позначки, що забезпечує розмежування доступу до інформації про об’єкт логістики.

Основні результати дисертаційної роботи опубліковано у 7 наукових працях, зокрема у 5 наукових статтях, з яких 1 статтю опубліковано у закордонному фаховому виданнях третього квартиля (Q3), яке проіндексоване в базі даних Scopus, 1 статтю опубліковано у періодичному науковому виданні держави, яка входить до Організації економічного співробітництва та розвитку та Європейського Союзу, 1 статтю опубліковано у виданні, включеному до переліку наукових фахових видань України з присвоєнням категорії «А», і 2 статті опубліковано у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б», та у 2 матеріалах науково-технічних конференцій, з яких 1 публікація у матеріалах міжнародної наукової конференції, що проіндексовано у базі даних Scopus.

Ключові слова: прикладне програмне забезпечення, архітектура програмного забезпечення, шаблон проєктування, логістична програмна система, автоматична ідентифікація об’єктів, штрихове кодування.

SUMMARY

Sulema O. Algorithms and software for logistics object automatic identification processes based on barcodes with three gradations of colour. − Qualifying scientific work, the manuscript.

PhD thesis in the field of knowledge 12 Information technologies in a specialty 121 Software engineering. – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2021.

Automatic object identification is an important aspect in many areas of human activity, one of which is logistics. The use of technologies for automatic identification of logistics objects makes it easier to track their location during transportation and warehousing, as well as to ensure the processes of control, reporting, etc. This has a positive effect on the intensification of production, trade, including international, postal services, etc. due to error-free and fast access to information about accounting objects, as well as automatic collection of data about them and updating this data in software systems without manual intervention.

Automatic object identification can be based on barcode data technologies, radio frequency identification, smart cards, etc. One of the most common technologies for automatic object identification is barcoding technology, which provides high accuracy and speed of information entry into computer systems and is economically attractive due to low cost of consumables and affordable equipment.

Nowadays, there is a number of approaches to the presentation of data in the form of barcodes, but increasing both the density of information in the form of barcodes and the noise immunity of barcodes still remains a topical task. Another important problem related to the creation of software systems based on barcoding technology for the logistics industry is the lack of a problem-oriented approach to integrating elements of barcoding technology in the development of software for automatic object identification processes. The presence of these

problems determines the topical scientific and technical problem of improving the development efficiency of algorithmic methods and software for automatic identification of objects based on barcodes with three colour gradations, which is solved in this dissertation.

The purpose of the dissertation is to increase the efficiency of logistic systems by automating production processes using high-dense data barcoding of logistics objects.

The first section of the dissertation analyses algorithmic and software solutions for automatic identification of objects in the field of logistics, in particular, the current state of logistics software, methods of automatic identification of objects and features of barcode information processing, which allowed to form basic requirements for software of automatic identification of logistics objects.

In the second section the algorithmic methods and software for processes of data formation on the basis of barcodes with three gradations of colour are developed, in particular, the problem of providing compact representation of data in the form of barcode images is formulated, the algorithmic methods and software for data consolidation for formation of barcodes and the algorithmic methods and software for formation of two-layer barcode symbols are developed and investigated.

In the third section the algorithmic methods and software of noise immunity of barcodes with three colour gradations are developed, in particular, the problem of ensuring noise immunity of barcode images is formulated, the algorithmic methods and software of noise immunity of barcodes with three gradations of colour are developed and investigated.

The fourth section is devoted to the creation of the design technology elements for logistics objects automatic identification software based on barcodes with three colour gradations, in particular, a scale of requirement prioritization for software systems development is proposed, the requirements

for a software system for logistics objects automatic identification based on barcodes with three colour gradations are formulated, the basic architecture of a software system for objects automatic identification based on barcodes with three colour gradations and a design pattern «Converter» are proposed, and the main stages of design software for logistics objects automatic identification based on barcodes codes with three colour gradations are formulated.

The dissertation provides a number of **new scientific results**, in particular, the basic architecture of a software system for automatic identification of logistics objects based on barcodes with three gradations of colour is proposed **for the first time**; the use of this architecture simplifies the process of software development for automatic object identification systems; unlike the existing ones, this architecture allows to provide two levels of access to information about the logistics object and to increase the density of data representation through the use of two-layer barcodes with three gradations of colour.

For the first time, the structural method of compression of alphanumeric data to be presented in the form of a barcode with three gradations of colour is developed; the method provides the use of several additional alphabets with capacities less than 256 (ASCII power), in addition to the computer alphabet ASCII; the method is based on splitting the input alphanumeric sequence into adjacent subsequences of characters and converting them into tricolour (triple) barcodes, the total length of which is less than the triple length of incoming messages, which increases the information density of data on a carrier in 1.16-1.5 times on average and prevails similar characteristics in the case of using statistical methods (based on Huffman, Shannon-Fano codes) for data compression.

For the first time, a mathematical model, which allows selecting an optimal number of additional (except ASCII) alphabets for converting input alphanumeric data to be presented in the form of barcodes with three gradations of colour, is developed; this enables determination of both an optimal power of

these alphabets and a type of character subsequence conversion into a numerical form that allow to achieve the maximum possible information compression of data to be located on a barcode carrier.

For the first time, the algorithmic methods for noiseless barcoding with three gradations of colour is proposed; its defining feature is the fulfilment of calculations in a finite field $GF(3^s)$ by a modulo of an irreducible polynomial of power s that allows to correct distortions of two types – errors and erasures, and enables data recovering when up to 37.5% of the area of the barcode is damaged.

For the first time, the two-layer barcoding method that allows to encode two independent sets of data, as well as the algorithms for forming two-layer barcodes with three gradations of colour, which distinctive feature is the use of specific procedures for determining a control bit for encoding the second data set and making a two-layer barcode that provides a delimitation of access to the information, are developed.

The main results of the dissertation **were published** in 7 scientific papers, in particular, in 5 scientific articles, including 1 article published in a foreign scientific journal of the third quartile (Q3), which is indexed in the Scopus database, 1 article published in a scientific journal of the country, which is a part of the Organization for Economic Cooperation and Development and the European Union, 1 article published in a scientific journal included in the list of scientific journals of Ukraine in category «А», and 2 articles published in scientific journals included in the list of scientific journals of Ukraine in category «Б», as well as in 2 materials of scientific and technical conferences, including 1 paper in the proceedings of the international scientific conference indexed in Scopus database.

Keywords: application software, software architecture, software design pattern, logistics software system, automatic object identification, barcoding.

Список публікацій здобувача / List of publications of the applicant:

Стаття у закордонному фаховому виданні третього квартиля (Q3), яке проіндексоване в базі даних Scopus / the article published in a foreign scientific journal of the third quartile (Q3), which is indexed in the Scopus database:

1. Dychka, I., Sulema, O., Salenko, A., Sulema, Y. Augmented Reality Application Based on Information Barcoding. Advances in Intelligent Systems and Computing. Springer Nature Switzerland AG. 2021. Vol. 1192, P. 750–761. ISSN : 21945357.

Стаття у періодичному науковому виданні держави, яка входить до Організації економічного співробітництва та розвитку та Європейського Союзу / the article published in a scientific journal of the country, which is a part of the Organization for Economic Cooperation and Development and the European Union:

2. Ivan Dychka, Olga Sulema. Black-Gray-White Barcode Based on Error Correction Data Encoding. Journal of Applied Computer Science. Poland, Lodz, 2019. Vol. 27, No. 2. P. 17–38. ISSN : 1507-0360.

Стаття у виданні, включеному до переліку наукових фахових видань України з присвоєнням категорії «А» / the article published in a scientific journal included in the list of scientific journals of Ukraine in category «A»:

3. Dychka I., Onai M., Sulema O. Data Compression in Black-Gray-White Barcoding. Radio Electronics, Computer Science, Control, 2020. No 1, P. 125–134.

Статті у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б» / the articles published in

scientific journals included in the list of scientific journals of Ukraine in category «Б»:

4. Onai M.V., Sulema O.K., Dychka A.I. Data Encoding Based On Tricolor Matrix Barcodes. KPI Science News, 2019. Vol. 2, P. 37–45.
5. І. А. Дичка, О. К. Сулема, А. А. Крайносвіт. Програмна система логістичного обліку на основі дворівневого штрихового коду. Системні технології, 2020. № 6, С. 28–38.

Публікація у матеріалах міжнародної наукової конференції, що проіндексовані у базі даних Scopus / the publication in the proceedings of the international scientific conference indexed in Scopus database:

6. Dychka, I., Sulema, O. Data compression and representation as multicolor barcodes. CEUR Workshop Proceedings, 2019. Vol. 2393, P. 534–541.

Публікація у матеріалах наукової конференції / the publication in the proceedings of the scientific conference:

7. Сулема О.К. Спосіб ущільнення даних при поданні інформації у вигляді триколірних двовимірних штрихових кодів. Десята наукова конференція магістрантів та аспірантів, Київ, 21–23 березня 2018 р. Збірник тез доповідей. Нац. техн. ун-т України «Київ. політехн. ін-т ім. Ігоря Сікорського». Київ : Просвіта, 2018. С. 268–272. ISBN 978-617-7010-14-1.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ASCII – American Standard Code for Information Interchange (американський стандартний код обміну інформацією)

DMS – Distribution Management System (система управління дистрибуцією)

EAN – European Article Number (європейський номер артикулу)

ERP – Enterprise Resource Planning (планування ресурсів підприємства)

FMS – Freight Management System (система управління вантажо-перевезеннями)

GTIN – Global Trade Item Number (глобальний номер товарної позиції)

HCCB – High Capacity Color Barcode (кольоровий штриховий код високої ємності)

HCC2D – High Capacity Colored 2-Dimensional Code (кольоровий двовимірний код високої ємності)

ISBN – International Standard Book Number (міжнародний стандартний книжковий номер)

ISSN – International Standard Serial Number (міжнародний стандартний серійний номер)

LDPC – Low-Density Parity-check Code (код з малою щільністю перевірок на парність)

MRP – Material Resource Planning (планування матеріальних ресурсів)

QR-code – Quick Response code (код швидкого відгуку)

RFID – Radiofrequency Identification (радіочастотна ідентифікація)

SCM – Supply Chain Management (управління ланцюгами поставок)

SSCC – Serial Shipping Container Code (серійний код контейнерної доставки)

TMS – Transportation Management System (система управління транспортом)

WMS – Warehouse Management System (система управління складським господарством)

ДДШК – дворівневий двовимірний штриховий код

ДШК – двовимірний штриховий код

Мод – модуль, одиниця виміру довжини у штрихових кодах

Мод^2 – одиниця площі штрихкодового елемента

ШК – штриховий код

ВСТУП

Актуальність теми. Автоматична ідентифікація об’єктів є важливим аспектом у багатьох галузях людської діяльності, однією з яких є логістика. Застосування технологій автоматичної ідентифікації об’єктів обліку у логістиці дозволило досягти суттєвих позитивних змін у загальному процесі управління ланцюгами поставок, що позитивно впливає на інтенсифікацію виробництва, торгівлі, зокрема міжнародної, поштових сервісів тощо завдяки безпомилковому та швидкому доступу до інформації про об’єкти обліку, а також автоматичному збору даних про них та оновлення цих даних у програмних системах без втручання людини. Прикладами технологій автоматичної ідентифікації об’єктів є технології на основі штрихових кодів, радіочастотної ідентифікації, смарт-карт тощо.

Однією з найпоширеніших технологій автоматичної ідентифікації об’єктів є технологія на основі штрихового кодування інформації, яка забезпечує високу точність та швидкість введення інформації до комп’ютерних систем та є економічно привабливою завдяки низькій вартості витратних матеріалів і доступному обладнанню (принтер, сканер).

Незважаючи на наявність достатньо широкого спектру підходів до подання даних у вигляді штрихових кодів, досі залишаються актуальними задачі підвищення щільності подання інформації у вигляді штрихових кодів та підвищення завадостійкості штрихкодових позначок.

Іншою важливою проблемою, пов’язаною зі створенням програмних систем на основі технології штрихового кодування, зокрема для галузі логістики, є відсутність системного підходу до задачі інтегрування елементів технології штрихового кодування при розробленні програмного забезпечення процесів автоматичної ідентифікації об’єктів, оскільки використання технології штрихового кодування у системах логістичного обліку здебільшого обмежується використанням кодів EAN для кодування

одиниць товару, проте потенціал технології штрихового кодування є набагато більшим, оскільки застосування штрихових кодів дозволяє не лише компактно подавати інформацію у машиночитаному вигляді для автоматичного збору інформації, а й обмежувати доступ до інформації про об’єкти обліку.

Таким чином, наявність зазначених актуальних задач визначає актуальну науково-технічну задачу вдосконалення теоретичних та практичних основ розроблення алгоритмічного та програмного забезпечення процесів автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору, яка вирішується у цій дисертаційній роботі.

Зв’язок роботи з науковими програмами, планами, темами. Дослідження за темою дисертаційної роботи провадилось у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» в рамках виконання держбюджетних науково-дослідних робіт «Розроблення та дослідження методів оброблення, розпізнавання, захисту та зберігання медичних зображень в розподілених комп’ютерних системах» (номер державної реєстрації 0117U004267) та «Математичні та програмні методи оброблення мультимодальних даних моніторингу медико-біологічних об’єктів для діагностики стану здоров’я пацієнтів» (номер державної реєстрації 0120U102134).

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення ефективності логістичних систем за рахунок автоматизації виробничих процесів на основі високощільного штрихового кодування даних про об’єкти логістики.

Відповідно до поставленої мети основними задачами дослідження є:

- аналіз програмного забезпечення логістичних систем;

- аналіз методів штрихового кодування даних з точки зору особливостей їх програмної реалізації;
- аналіз вимог до програмних систем автоматичної ідентифікації об’єктів;
- розроблення базової архітектури програмної системи автоматичної ідентифікації об’єктів;
- розроблення та дослідження алгоритмічного і програмного забезпечення для штрихового кодування даних на основі штрихових кодів з трьома градаціями кольору;
- розроблення шаблону проєктування та визначення елементів технології проєктування програмних систем автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору.

Об’єкт дослідження – процеси розроблення програмних систем автоматичної ідентифікації об’єктів.

Предмет дослідження – методи розроблення програмного забезпечення систем автоматичної ідентифікації об’єктів на основі штрихових кодів.

Методи дослідження: теорія програмних систем, теорія програмування, теорія алгоритмів, теорія кодування, методи оптимізації.

Наукова новизна одержаних результатів полягає у наступному:

1. **Уперше** запропоновано базову архітектуру програмної системи автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, використання якої дозволяє спростити процес розроблення програмного забезпечення систем автоматичної ідентифікації об’єктів та яка, на відміну від існуючих, створює можливість дворівневого доступу до інформації про об’єкт логістики та підвищення щільності подання даних завдяки використанню дворівневих штрихових кодів з трьома градаціями

кольору.

2. **Уперше** розроблено структурний метод ущільнення алфавітно-цифрових даних, що підлягають поданню у вигляді штрихового коду з трьома градаціями кольору, який передбачає використання, крім комп'ютерного алфавіту ASCII, кількох додаткових алфавітів з потужностями, меншими за 256 (потужність ASCII), та ґрунтується на розбитті вхідної алфавітно-цифрової послідовності на суміжні підпослідовності символів та їх перетворенні у триколірні (трійкові) штрихкодові знаки, сумарна довжина яких менша, аніж трійкова довжина вхідних повідомлень, що підвищує інформаційну щільність подання даних на носії у середньому в 1,16–1,5 разів та переважає аналогічний показник у разі застосування для ущільнення статистичних методів (на основі кодів Хаффмана, Шеннона-Фано).
3. **Уперше** розроблено математичну модель, яка дозволяє здійснювати вибір оптимальної кількості додаткових (окрім ASCII) алфавітів для використання при перетворенні вхідних алфавітно-цифрових даних, що підлягають поданню у вигляді штрихових кодів з трьома градаціями кольору, та визначати оптимальні потужності цих алфавітів і тип перетворення підпослідовностей суміжних символів у числову форму, за яких досягається максимально можливе інформаційне ущільнення даних на штрихкодовому носії.
4. **Уперше** розроблено алгоритмічне забезпечення процесу завадозахищеного кодування штрихкодових позначок з трьома градаціями кольору, визначальною рисою якого є застосування обчислень у скінченому полі $GF(3^s)$ за модулем незвідного многочлена степеня s, що забезпечує виправлення спотворень двох видів – помилок і стирань, та уможливлює відновлення даних при ушкодженні до 37,5% площі штрихкодового зображення.

5. **Уперше** розроблено метод дворівневого штрихового кодування двох незалежних наборів даних та алгоритмічне забезпечення процесів формування штрихкодових позначок з трьома градаціями кольору, характерною рисою яких є застосування процедур визначення контрольного біта для подання другого набору даних та створення дворівневої штрихкодової позначки, що забезпечує розмежування доступу до інформації про об'єкт логістики.

Практичне значення одержаних результатів полягає у спрощенні процесу розроблення програмних систем автоматичної ідентифікації об'єктів на основі штрихових кодів шляхом використання шкали пріоритизації вимог до розроблюваної програмної системи, адаптування базової архітектури програмної системи до вимог та застосування шаблону проєктування «Перетворювач» при розроблені програмного коду.

Запропоновані алгоритмічне та програмне забезпечення подання даних у вигляді штрихових кодів з трьома градаціями кольору застосовані при виконанні науково-дослідних робіт «Розроблення та дослідження методів обробки, розпізнавання, захисту та зберігання медичних зображень в розподілених комп'ютерних системах» (номер держреєстрації 0117U004267) та «Математичні та програмні методи оброблення мультимодальних даних моніторингу медико-біологічних об'єктів для діагностики стану здоров'я пацієнтів» (номер державної реєстрації 0120U102134) для забезпечення компактного подання та швидкого і безпомилкового введення даних до комп'ютерних систем.

Особистий внесок здобувача. Всі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто. У публікаціях, написаних у співавторстві, здобувачеві належать наступні результати. У роботі [1] здобувачем запропоновано процедуру подання даних у вигляді триколірного штрихового коду. У роботі [2] здобувачем запропоновано алгоритмічне

забезпечення процесу подання даних у вигляді чорно-сіро-білого завадостійкого штрихового коду. У роботі [3] здобувачем запропоновано алгоритмічне забезпечення процесу подання даних у вигляді чорно-сіро-білого високощільного штрихового коду. У роботі [4] здобувачем запропоновано алгоритмічне забезпечення процесу подання даних у вигляді триколірного матричного штрихового коду. У роботі [5] здобувачем запропоновано метод дворівневого штрихового кодування з трьома градаціями кольору, шаблон проєктування програмного забезпечення «Перетворювач» і архітектуру логістичної програмної системи на основі технології штрихового кодування. У роботі [6] здобувачем запропоновано алгоритмічне та програмне забезпечення методу триколірного штрихового кодування.

Апробація результатів дисертації. Основні результати дисертаційного дослідження доповідалися та обговорювалися на міжнародних та національних науково-практичних конференціях:

1. 13th International Conference on Interactive Mobile and Communication Technologies and Learning 2019. Thessaloniki, Greece, 2019.
2. 15th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Kherson, Ukraine, 2019.
3. X конференція молодих вчених «Прикладна математика та комп'ютинг (ПМК-2018)». Київ, Україна, 2018.

Публікації. Основні результати дисертаційної роботи опубліковано у 7 наукових працях, зокрема у 5 наукових статтях, з яких 1 статтю опубліковано у закордонному фаховому виданнях третього квартиля (Q3), яке проіндексоване в базі даних Scopus, 1 статтю опубліковано у періодичному науковому виданні держави, яка входить до Організації економічного співробітництва та розвитку та Європейського Союзу, 1 статтю опубліковано у виданні, включеному до переліку наукових

фахових видань України з присвоєнням категорії «А», і 2 статті опубліковано у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б», та у 2 матеріалах науково-технічних конференцій, з яких 1 публікація у матеріалах міжнародної наукової конференції, що проіндексовано у базі даних Scopus.

Структура роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, 10 додатків та списку літератури, що включає 151 найменування. Загальний обсяг роботи становить 294 сторінки, у тому числі 150 сторінок основного тексту, 16 рисунків, 17 таблиць.

РОЗДІЛ 1. АНАЛІЗ АЛГОРИТМІЧНО-ПРОГРАМНИХ РІШЕНЬ ДЛЯ АВТОМАТИЧНОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ ЛОГІСТИКИ

1.1. Сучасний стан програмного забезпечення у галузі логістики

Автоматична ідентифікація об'єктів є важливим аспектом у багатьох галузях людської діяльності, однією з яких є логістика [6, 27, 40, 90, 91, 97, 106, 107, 111–114, 139]. Застосування технологій автоматичної ідентифікації об'єктів обліку у логістиці дозволило досягти суттєвих позитивних змін у загальному процесі управління ланцюгами поставок, що позитивно впливає на інтенсифікацію виробництва, торгівлі, зокрема міжнародної, поштових сервісів тощо завдяки безпомилковому та швидкому доступу до інформації про об'єкти обліку, а також автоматичному збору даних про них та оновлення цих даних у програмних системах без втручання людини.

Основною задачею, що вирішується за допомогою логістичних програмних систем з автоматичною ідентифікацією об'єктів, є забезпечення всіх учасників логістичних процесів – виробників продукції, постачальників логістичних послуг та споживачів логістичних послуг (наприклад, торгівельні організації) – точною та актуальною інформацією про поточне місцезнаходження та рух одиниці логістичного обліку. Виявлення, збір та оперативне надання цієї інформації разом з цифровізацією логістичних бізнес-процесів не лише покращує взаємодію між зацікавленими сторонами, але й створює високоефективне, стійке та спільне логістичне середовище, що сприяє підвищенню загальної якості логістичних послуг, а отже, й економічний ефект від їх надання.

Сучасне програмне забезпечення у галузі логістики [4, 37, 43, 46, 51, 55, 86, 93] характеризується наступними властивостями.

1. Підтримка ланцюгів поставок (SCM-системи), які підтримують весь спектр логістичних послуг.
2. Забезпечення автоматичної ідентифікації об’єктів логістичного обліку за допомогою технологій штрихового кодування [1, 2, 7, 9, 19, 26, 32, 35, 44, 50, 94, 98, 108, 110, 115–138, 140–150], смарт-карток, RFID [30, 67, 71, 73, 109].
3. Забезпечення автоматичного трекінгу при транспортуванні об’єктів логістичного обліку.

Проте аналіз наявних логістичних програмних систем [4, 37, 43, 46, 51, 55, 86, 93] показує, що використання технологій автоматичної ідентифікації обмежене ідентифікацією самих об’єктів логістики, що реалізується шляхом нанесення спеціальної машиночитаної етикетки на одиницю обліку. Водночас, можливості, які надає штрихове кодування, для автоматичного збору даних та автоматизованого введення інформації з машиночитаних документів (документів, які містять інформацію, що подана виключно чи переважно у штрихкодованому вигляді), практично не використовуються.

У галузі логістики розрізняють наступні види програмних систем (табл. 1.1).

1. Системи управління дистрибуцією (DMS). Забезпечують планування закупівель товарів. Передбачається використання технології автоматичної ідентифікації товарів, зокрема технології штрихкодової ідентифікації.
2. Системи управління вантажоперевезеннями (FMS). Забезпечують планування, замовлення та відстеження вантажоперевезень. Передбачається використання технології автоматичної ідентифікації вантажів, зокрема технології штрихкодової ідентифікації та технології радіочастотної ідентифікації.

3. Системи управління транспортом (TMS). Забезпечують планування, виконання та оптимізацію перевезень. Мають схожі функції з FMS.
4. Системи управління складським господарством (WMS). Забезпечують реєстрацію, складування, зберігання та облік товарів й інших об’єктів. Передбачається використання технології автоматичної ідентифікації товарів, зокрема технології штрихкодової ідентифікації.
5. Системи управління ланцюгами поставок (SCM). Забезпечують повний цикл поставок товарів. Включають системи інших типів як підсистеми.

Таблиця 1.1. Порівняння програмних систем, що використовуються у галузі логістики

Назва	Тип	Відкритість архітектури	Орієнтація на використання хмарних сховищ
Magaya	SCM	Так	Так
iSupply	FMS, DMS, WMS	Сумісність з ERP, MRP системами	Так
Rackbeat	WMS	Так	Так
Softeon	WMS, TMS	Сумісність з ERP, MRP системами	Так
Kuebix	FMS, TMS	Сумісність з ERP, MRP системами	Так
CargoWise	WMS, TMS	Сумісність з ERP, MRP системами	Так
KeepTruckin	TMS	Сумісність з ERP, MRP системами	Так

Порівняльний аналіз найвідоміших програмних систем, що використовуються у галузі логістики (табл. 1.1) показує, що частина цих систем підтримують розширення функціональності, оскільки мають

відкриту архітектуру та орієнтовані на використання хмарних сховищ для обміну даними, тому розроблення нового програмного забезпечення, яке орієнтоване на розширене використання технологій автоматичної ідентифікації об’єктів, зокрема технології штрихового кодування, є доцільним, а задача створення нових вдосконалених програмних засобів для автоматичної ідентифікації об’єктів логістики є актуальною.

Важливим компонентом розглянутих програмних систем є модуль автоматичної ідентифікації об’єктів логістики. Розглянемо докладно методи автоматичної ідентифікації об’єктів, які на сьогодні використовуються для розроблення програмного забезпечення логістичних систем.

1.2. Аналіз методів автоматичної ідентифікації об’єктів

Автоматична ідентифікація об’єктів може ґрунтуватись на технологіях на основі штрихових кодів, радіочастотної ідентифікації, смарт-карт тощо. При виборі тієї чи іншої технології автоматичної ідентифікації об’єктів для певної галузі застосування потрібно брати до уваги багато чинників, включаючи доступність та економічну ефективність. За цими критеріями найпривабливішою технологією упродовж багатьох років залишається технологія штрихового кодування. Це пояснюється не лише низькою вартістю обладнання та витратних матеріалів, але й високою точністю і швидкістю введення даних, а також можливістю використання звичайних технічних засобів (принтер, сканер) для застосування цієї технології.

За час існування технології штрихового кодування [1, 2, 7, 9, 19, 26, 32, 35, 44, 50, 94, 98, 108, 110, 115–138, 140–150] було розроблено десятки штрихових кодів (ШК), які можуть бути класифіковані відповідно до наступних категорій.

1. За вимірністю:
   - одновимірні (лінійні);
   - двовимірні (стекові, матричні).
2. За формою штрихкодової позначки:
   - прямокутні, квадратні;
   - кругові, радіальні, кільцеві;
   - різновеликі.
3. За формою елементів штрихкодової позначки:
   - на основі прямокутних елементів;
   - на основі елементів у вигляді відрізків прямої (штрихів);
   - на основі елементів у вигляді точок;
   - на основі шестикутних елементів.
4. За колірністю:
   - монохромні;
   - кольорові.

Найпоширенішими на сьогодні є лінійні (прямокутні) монохромні коди сімейства EAN / UPC [19, 134], які переважно використовуються у торгівлі, та матричні (квадратні) монохромні коди QR [35, 107], які використовуються для подання алфавітно-цифрових кодів, зокрема з метою надання швидкого доступу до онлайн-ресурсів.

Розглянемо та проаналізуємо ці та інші поширені способи машиночитаного кодування.

1.2.1. Лінійні штрихові коди сімейства EAN / UPC

Штрихові коди сімейства EAN / UPC [19, 134] є найбільш відомими та поширеними ШК. Вони були створені для галузі торгівлі для швидкої ідентифікації товарів у пунктах продажу, проте їх застосування є дещо ширшим.

До штрихових кодів сімейства EAN / UPC належать коди EAN-8, EAN-13, EAN-14, EAN-18, EAN-99, EAN-128, UPC-A, UPC-E, додаткові коди EAN-2 та EAN-5, а також коди, які є похідними від кодів EAN, зокрема коди ISBN та ISSN.

Найбільш поширеним серед кодів сімейства EAN / UPC є код EAN-13 або GTIN-13, який в основному використовується в супермаркетах для ідентифікації товару в місці продажу. ШК-позначка коду EAN-13 подає номер EAN або GTIN для ідентифікації одиниці товару.

Код EAN-8 або GTIN-8 є короткою формою коду EAN-13. Цей код використовується, лише якщо об’єкт обліку за розміром є замалим для позначки з кодом EAN-13, а саме, якщо позначка з кодом EAN-13 займає більше 25% поверхні об’єкту обліку.

Код EAN-99 є спеціальною формою EAN-13, що починається з префіксу «99». Призначенням ШК EAN-99 є використання як купону у торгівельних підприємствах. Зазвичай купони з ШК EAN-99 розповсюджуються у магазині, в якому вони будуть використані.

Код UPC-A або GTIN-12 є аналогічним коду EAN-13, проте він дозволяє подавати код товару, що складається з 12 цифр.

Код UPC-E є короткою формою коду UPC-A. Цифрова послідовність, що подається у вигляді ШК-позначки коду UPC-E, має завжди починатися з префіксу «0».

Код EAN-128 або GS1-128 використовується для подання даних про одиниці обліку у торгівлі та промисловості. ШК позначка EAN-128 містить GTIN, а також може містити конкретну інформацію про товар (вага, дата виготовлення, серійний номер, термін придатності тощо).

Код EAN-14 або GTIN-14 призначено для кодування товарів роздрібної торгівлі. ШК-позначка EAN-14 створюється з використанням символіки EAN-128.

Код EAN-18 або SSCC-18 використовується по всьому ланцюжку поставок як ідентифікатор для відстеження товару та внутрішнього контролю. ШК-позначка EAN-18 створюється з використанням символіки EAN-128.

Код ISSN використовується для ідентифікації періодичних видань, зокрема журналів. ШК ISSN створюється за допомогою символіки EAN-13 з префіксом «977».

Код ISBN або Bookland EAN-13 використовується для ідентифікації книжок. ШК ISBN створюється за допомогою символіки EAN-13 з префіксом «978» або «979».

Коди EAN-5 та EAN-2 є додатковими кодами, що призначені для використання разом з ШК EAN-13 або UPC-A. Додатковий код EAN-2 призначений для кодування додаткових даних (ціна) для розміщення у газетах та журналах, а додатковий код EAN-5 призначений для розміщення на обкладинках книжок. Обидва коди використовуються лише як додаток до кодів EAN-13, EAN-8 та UPC.

Будову ШК-позначки та порівняння розглянутих вище кодів наведено у Додатку А, зокрема порівняльну характеристику зазначених кодів наведено у табл. А.1.

Принцип визначення контрольного значення є схожим для всіх кодів сімейства EAN / UPC, в яких передбачене використання контрольного значення, проте використовуються три різновиди алгоритму додавання контрольного значення.

1. Алгоритм за модулем 10. Використовується для всіх кодів сімейства EAN / UPC, крім кодів ISSN та EAN-128.
2. Алгоритм за модулем 11. Використовується для кодів ISSN.
3. Алгоритм за модулем 103. Використовується для кодів EAN-128.

Алгоритм додавання контрольного значення за модулем 10 представлено на рис. 1.1. У наведеній блок-схемі цього алгоритму використовуються наступні позначення: $d_1 d_2 \ldots d_N$ – вхідна цифрова послідовність довжини *N*; *Index* – поточний індекс; *S* – зважена сума; *k* – контрольне значення.

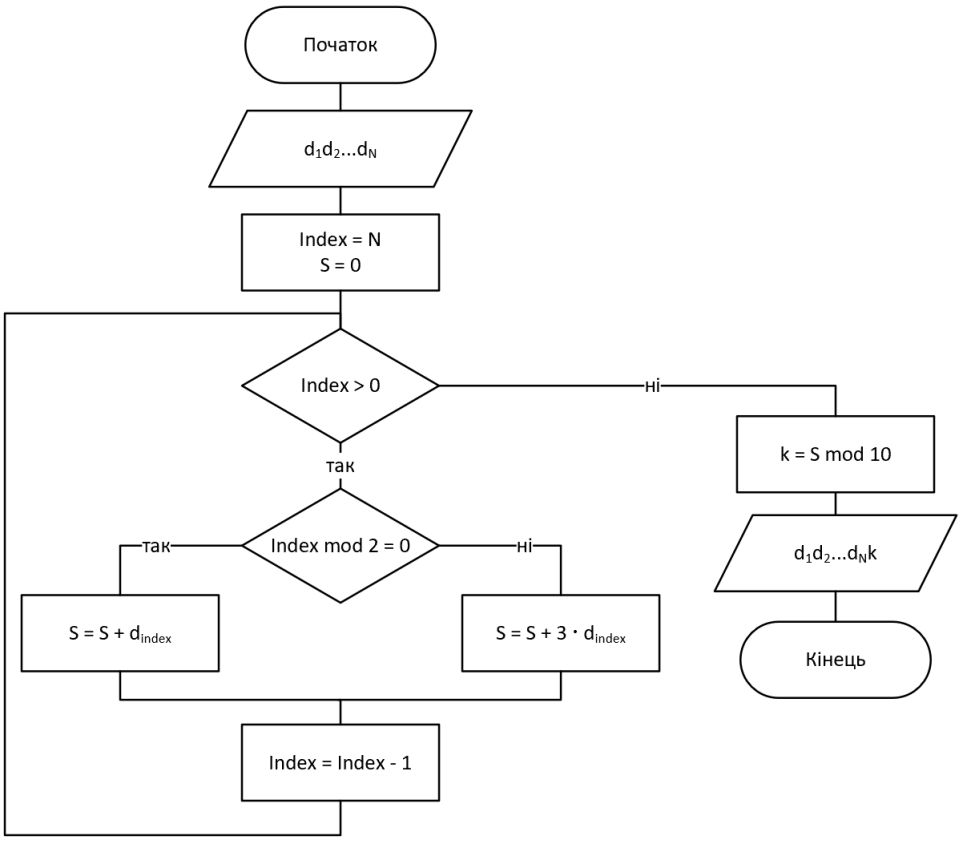


Рис. 1.1. Алгоритм додавання контрольного значення за модулем 10

Алгоритм додавання контрольного значення за модулем 11 представлено на рис. 1.2. У наведеній блок-схемі цього алгоритму використовуються наступні позначення: $c_1 c_2 \ldots c_N$ – вхідна послідовність довжини *N*, що складається з цифр та символу «X»; *Index* – поточний індекс; *S* – зважена сума; *k* – контрольне значення; *SymbolConversion* – процедура перетворення вхідного символу на цифрове значення;

ValueConversion – процедура перетворення цифрового значення на вихідний символ.

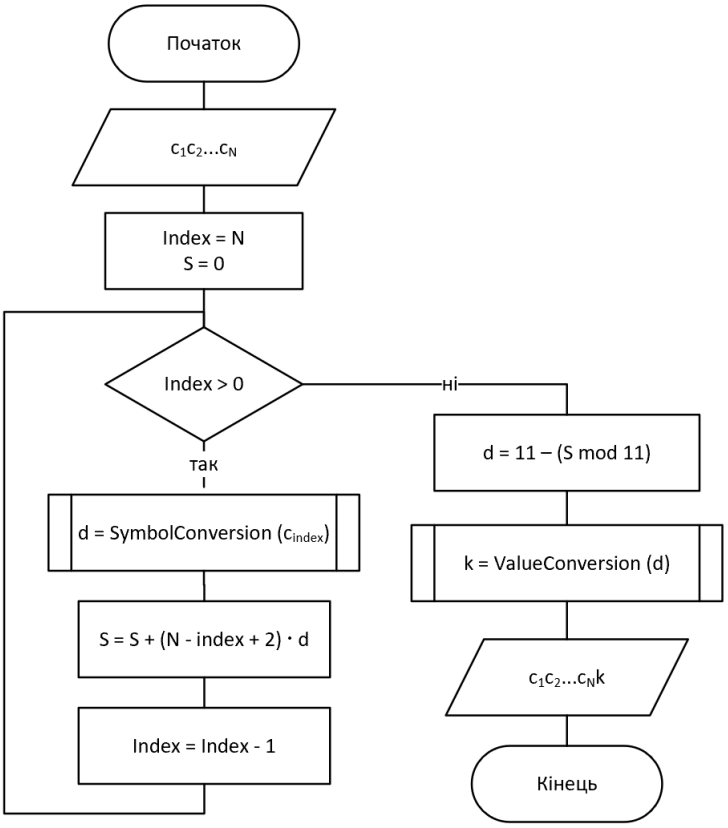


Рис. 1.2. Алгоритм додавання контрольного значення за модулем 11

Алгоритм додавання контрольного значення за модулем 103 представлено на рис. 1.3. У наведеній блок-схемі цього алгоритму використовуються наступні позначення: $a_1a_2\ldots a_N$ – вхідна послідовність довжини *N*, що складається з символів таблиці ASCII; *b* – префіксний символ, що визначає тип набору вхідних символів; *Index* – поточний індекс; *S* – зважена сума; *k* – контрольне значення; *ASCII_SymbolConversion* – процедура перетворення вхідного символу з таблиці ASCII на цифрове значення; *ASCII_ValueConversion* – процедура перетворення цифрового значення на вихідний символ з таблиці ASCII.

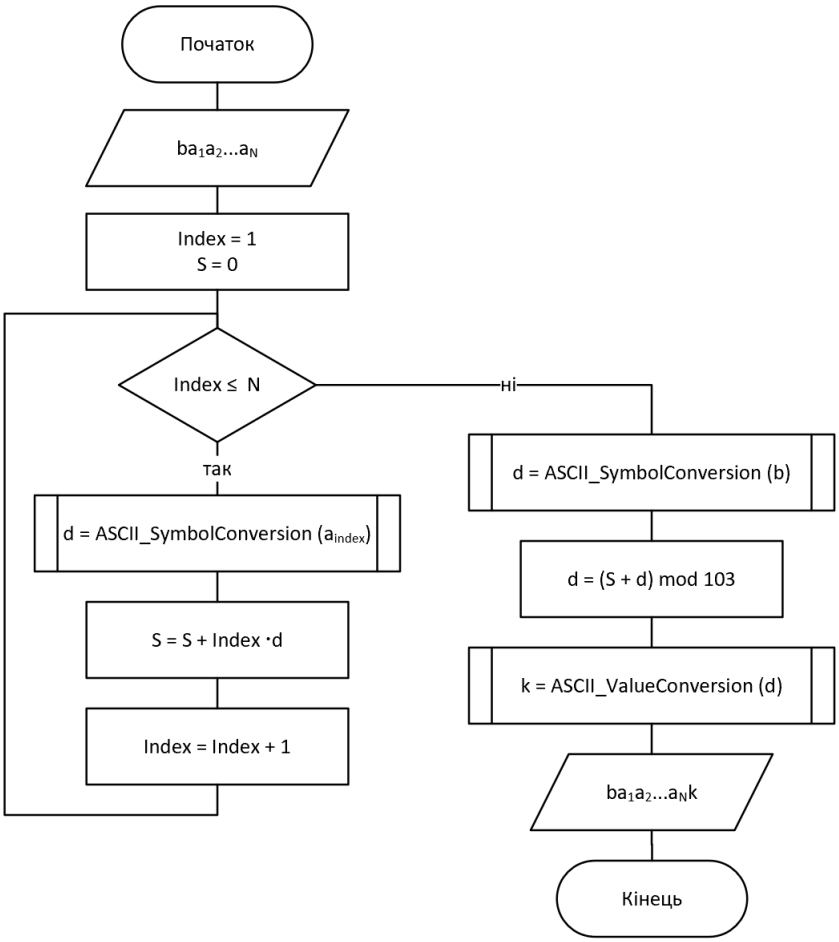


Рис. 1.3. Алгоритм додавання контрольного значення за модулем 103

Аналіз даних у табл. 1.2 та алгоритмів додавання контрольного значення дозволяє зробити висновок про невелику інформаційну ємність та низьку завадостійкість ШК-позначок кодів сімейства EAN / UPC.

1.2.2. QR-код

QR-код [35] є матричним монохромним кодом, який застосовується для подання даних для забезпечення швидкого доступу до онлайн-ресурсів. До сімейства матричних кодів QR, окрім QR-коду, можна віднести його модифікації: MicroQR, iQR Code, FrameQR, SQRC. Оскільки серед зазначених кодів найбільше поширення здобув QR-код, розглянемо його докладно.

QR-код має наступні важливі властивості [35].

1. Масштабованість. Передбачається можливість використання 40 версій позначок: від позначки мінімального розміру 21×21 комірок до позначки максимального розміру 177×177 комірок. Крок зміни розміру позначки складає 4 комірки, тобто можливе формування позначок з стороною 21, 25, 29, …, 173, 177 комірок.
2. Наявність великого алфавіту. Передбачається наявність чотирьох наборів символів: цифри, алфавітно-цифрові символи, бінарні значення, символи японської ієрогліфічної системи запису Кандзі.
3. Завадостійкість. Передбачається можливість використання чотирьох рівнів завадостійкості: Level L, Level M, Level Q, Level H.

Основні характеристики QR-коду наведено у табл. Б.1–Б.3.

При використанні позначки максимального розміру, 177×177 комірок, що відповідає версії 40 стандарту QR-коду, максимальна ємність позначки складає 7089 символи за умови використання цифрового набору даних.

Інформаційна щільність подання даних у вигляді позначки QR-коду залежить від рівня завадостійкості QR-коду, що застосовується. Як було зазначено вище, існує чотири рівні завадостійкості:

- Level L передбачає можливість відновлення даних, що спотворено до 7% загальної площі позначки;
- Level M передбачає можливість відновлення даних, що спотворено до 15% загальної площі позначки;
- Level Q передбачає можливість відновлення даних, що спотворено до 25% загальної площі позначки;
- Level H передбачає можливість відновлення даних, що спотворено до 30% загальної площі позначки.

Для забезпечення завадостійкості використовуються коди Ріда-Соломона. Наприклад, у разі використання позначки максимального

розміру, 177×177 комірок, та найвищого рівня завадостійкості, Level H, для кожного блока даних використовується код з параметрами $(c, k, r) = (46, 16, 15)$, де c – загальна кількість кодових слів, k – кількість інформаційних кодових слів, r – кількість контрольна кодових слів (табл. Б.2). У табл. Б.3 показано залежність інформаційної ємності від рівня завадостійкості для QR-коду максимального розміру.

Будову кодової позначки для розглянутих вище кодів наведено у Додатку Б.

Аналіз характеристик QR-коду, зокрема наведених у табл. Б.1–Б.3, дозволяє зробити висновок про високу інформаційну ємність та високу завадостійкість позначок QR-коду, проте обидві характеристики мають обмежене значення, зокрема за допомогою QR-коду неможливо подати більш, ніж 3 Кб даних. Крім того, особливістю позначок QR-коду є те, що їхній розмір є регламентованим стандартом та не може бути змінений. Іншою особливістю QR-коду є обмеженість адаптації вхідного алфавіту до потреб конкретного варіанта застосування.

Оскільки QR-код широко застосовується, то доцільно провести дослідження програмного забезпечення, яке дозволяє формувати та розпізнавати позначки QR-коду, щоб виявити практичні проблеми, які виникають під час застосування QR-коду. Результати цього дослідження представлені у табл. Б.4.

Отже, існує велика кількість програмних застосунків, які дозволяють працювати з позначками QR-коду, що підтверджує його широке використання та попит на відповідне програмне забезпечення. Водночас, проведене дослідження наявного програмного забезпечення для формування та розпізнавання QR-позначок наочно продемонструвало обмеженість обсягу інформації, яку можна подати у вигляді QR-коду.

1.2.3. Матричні кольорові коди

На сьогодні відомо декілька різновидів двовимірних кольорових кодів, а саме, JAB [94], HCCB, HCC2D [83–85]. Ці коди відрізняються кількістю кольорів, що використовуються, формою комірок та завадостійкістю.

Основні характеристики зазначених кодів наведено у табл. В.1.

Оскільки найдосконалішим з точки зору основних характеристик є код JAB, розглянемо його докладніше [94].

Код JAB використовує палітру з 8 базових кольорів, яким присвоєно відповідні номери, а саме: чорний (0), синій (1), зелений (2), блакитний (3), червоний (4), пурпурний (5), жовтий (6), білий (7). Ці базові кольори використовуються для отримання від чотирьох до 256 відтінків кольорів. Як зазначено у табл. 1.7, позначка коду JAB може мати 32 різних розміри. Найменша квадратна позначка має розміри 21 × 21 модуля, найбільша квадратна позначка має розмір 145 × 145 модулів. Найменша прямокутна позначка має розміри 21 × 25 модулів, найбільша прямокутна позначка має розмір 141 × 145 модулів. Максимальну пропорцію між горизонтальною та вертикальною сторонами мають прямокутні позначки 21 × 145 модулів або 145 × 21 модулів.

Символіка коду JAB включає 7 наборів даних.

1. Набір цифрових даних. Включає 10 цифр, проміжок, кому та крапку.
2. Набір літер верхнього реєстру та проміжок.
3. Набір літер нижнього реєстру та проміжок.
4. Набір знаків пунктуації.
5. Набір допоміжних символів (наприклад, сім літер німецької абетки – Ä, Ö, Ü, ä, ö, ü, ß).
6. Набір алфавітно-цифрових даних.
7. Набір бінарних значень.

У коді JAB передбачено використання двох типів позначок: первинна та вторинна, які мають різну структуру та можуть комбінуватися. Будову кодової позначки для розглянутих вище кодів наведено у Додатку В. Ємність даних коду JAB залежить від розміру позначки, кількості кольорів та рівня виправлення помилок, який визначається типом позначки. У табл. В.2 наведено дані про інформаційну ємність квадратної позначки коду JAB для деяких стандартних розмірів.

Завадостійкість позначки коду JAB забезпечується за допомогою коду з малою щільністю перевірок на парність – LDPC, який застосовується до бінарних даних. У коді JAB визначено 11 рівнів завадостійкості, які наведені у табл. В.3.

Аналіз характеристик коду JAB дозволяє зробити висновок про високу інформаційну ємність та високу завадостійкість позначок коду JAB. Проте, для використання коду JAB необхідно забезпечити високу якість друку та сканування позначок.

Вимога щодо забезпечення високої якості друку та сканування позначок є важливою й для застосування інших кольорових кодів. Розпізнавання кольорів може бути ускладнене неякісним друком, забрудненням об’єктиву камери портативного пристрою, дефектами позначки тощо. Розглянемо особливості процедур виготовлення та використання штрихкодових позначок, що є складовими штрихкодової ідентифікації об’єктів.

1.3. Аналіз особливостей оброблення штрихкодованої інформації для формування вимог до програмно-апаратного забезпечення процесів автоматичної ідентифікації об’єктів логістики

При проєктуванні програмних систем автоматичної ідентифікації об’єктів на основі технології штрихового кодування потрібно враховувати

особливості оброблення інформації, поданої у вигляді ШК-позначки. Зокрема, ці особливості мають бути враховані при формуванні вимог до програмно-апаратного забезпечення процесів автоматичної ідентифікації об’єктів логістики. Розглянемо основні особливості, пов’язані з формуванням, використанням та розпізнаванням ШК-позначок.

1.3.1. Особливості оброблення колірної інформації при формуванні та розпізнаванні ШК-позначок

Оброблення колірної інформації при формуванні та розпізнаванні ШК-позначок безпосередньо пов’язане зі способами подання інформації про відтінок кольору на основі певної колірної моделі [34] – формального способу визначення відтінку кольору.

Найвідомішою колірною моделлю є модель RGB, яка передбачає опис відтінку кольору шляхом визначення пропорції змішування базових кольорів, якими для цієї моделі є червоний, зелений та синій, для отримання потрібного відтінку кольору. Іншою моделлю, яка використовує аналогічний спосіб формального визначення відтінка кольору є колірна модель CMY, в якій базовими кольорами є блакитний, пурпурний та жовтий, та модель CMYK, яка є модифікацією моделі CMY та відрізняється від неї використанням чорного кольору як ще одного, четвертого, базового кольору.

У колірних моделях YCbCr, YUV та YIQ використовується інший підхід до формального визначення відтінка кольору, а саме: відтінок кольору визначається через значення яскравості (так званий, яскравісний канал Y) та два кольорорізнісні значення, спосіб визначення яких залежить від конкретної моделі.

У моделях HSV, HLS та HSI також відбувається відокремлення значення яскравості, проте, на відміну від моделей YCbCr, YUV та YIQ,

базовий відтінок кольору у моделях HSV, HLS та HSI задається явно як координата H у колірному просторі відповідної колірної моделі.

Розуміння способів формального визначення відтінку кольору дозволяє проаналізувати доцільність використання того чи іншого набору кольорів у кольорових ШК.

Як показують дані у табл. В.1, в існуючих ДШК використовують від 4 до 256 кольорів. Це робиться з метою підвищення інформаційної ємності ШК-позначки. Проте, застосування великої кількості кольорів ускладнює розпізнавання ШК-позначок. Для забезпечення надійного декодування штрихкодованих даних використовують методи класифікації кольорів [83–85]: класифікацію за методом мінімальної відстані, дерева рішень, кластеризацію за методом k-середніх, наївний баєсів класифікатор, метод опорних векторів тощо. Втім, використання зазначених методів класифікації кольорів підвищує обчислювальну складність програмного забезпечення автоматичної ідентифікації об’єктів.

Щоб уникнути підвищення обчислювальної складності програмних компонентів, що призначені для декодування ШК-позначок, та при цьому забезпечити високу інформаційну ємність ШК, доцільно обмежити кількість кольорів, що використовуватимуться для подання логістичної інформації. Монохромні (чорно-білі) та повнокольорові ШК (256-колірні) є діаметрально протилежними підходами до використання кольорів для графічного кодування даних у вигляді ШК.

Компромісним рішенням, яке пропонується у цьому дисертаційному дослідженні, є використання кількох градацій одного кольору, зокрема використання трьох градацій:

- мінімальної (білий колір),
- максимальної (основний колір, наприклад, чорний),
- середньої (напівтон основного кольору, наприклад, сірий).

Вибір основного кольору залежить від вимог до дизайну носія ШК-позначки та наявного обладнання для виготовлення і сканування ШК-позначки. Важливою вимогою при виборі основного кольору є забезпечення контрасту, достатнього для безпомилкового декодування даних.

У теорії кольорів контрастом називають міру різниці у яскравості між двома кольорами. Яскравість – це інтенсивність світла, що випромінюється з поверхні об’єкта на одиницю площі [96]. Як було зазначено вище, яскравість може бути окремим компонентом визначення кольору, що використовується у колірних моделях YCbCr, HSI тощо.

Контраст може бути визначений як коефіцієнт контрастності кольорів – відношення яскравості світлішого з двох кольорів (наприклад, білого) до яскравості темнішого з цих кольорів (наприклад, чорного) [92, 103]:

$$r_c = \frac{L_1 + 0.05}{L_2 + 0.05}, \tag{1.1}$$

де L_1 – відносна яскравість світлішого з двох кольорів, а L_2 – відносна яскравість темнішого з двох кольорів.

Для забезпечення безпомилковості процедури сканування ШК-позначки також важливо, щоб кольори, що використовуються у триколірному ШК, були контрастними до фону об’єкта-носія. Таким чином, кольори тла також мають бути проаналізовані з урахуванням ступеня контрасту перед тим, як створювати ШК-позначку, щоб її комірки були зафарбовані кольорами з високим коефіцієнтом контрастності до тла.

Однією з вимог до застосування технології штрихового завжди є мінімізація витрат на обладнання для виготовлення та розпізнавання ШК-позначок, тому доцільним є врахування у вимогах до апаратно-програмної

системи автоматичної ідентифікації об’єктів логістики можливість використання звичайної офісної техніки для друку та сканування ШК-позначок. Виходячи з цієї вимоги, доцільно обрати білий колір як найсвітліший колір та чорний колір як найтемніший колір.

Скористаємось формулою (1.1) та визначимо відносну яскравість третього кольору або в випадку, що розглядається, – певного відтінку чорного кольору, який забезпечитиме найкращий контраст ШК-позначки.

Тоді, мають виконуватись умови:

$$\begin{cases} \dfrac{L_{light} + 0.05}{L_{halftone} + 0.05} = \dfrac{L_{halftone} + 0.05}{L_{dark} + 0.05} \\ r_c = \dfrac{L_{light} + 0.05}{L_{dark} + 0.05} \to max \end{cases}, \quad (1.2)$$

де L_{light} – відносна яскравість світлого кольору, для білого кольору $L_{light} = 1$; L_{dark} – відносна яскравість темного кольору, для чорного кольору $L_{dark} = 0$; $L_{halftone}$ – відносна яскравість шуканого напівтону основного (темного) кольору.

Максимальне значення $r_c = 21$ можливе при застосуванні саме білого та чорного кольорів. Отже, знайдемо відтінок основного (темного) кольору, який задовольняє умовам (1.2).

$$\frac{1.05}{L_{halftone} + 0.05} = \frac{L_{halftone} + 0.05}{0.05},$$

$$\left(L_{halftone} + 0.05\right)^2 = 0.0525,$$

$$L_{halftone} = 0.179.$$

Таким чином, за третій колір у триколірному ШК потрібно обрати колір з відносною яскравістю 0.179. При знаходженні такого кольору доцільно взяти до уваги вимоги до дизайну ШК-позначки та умову, згідно формули відносної яскравості [103] вигляду:

$$L = 0.2126 * R + 0.7152 * G + 0.0722 * B, \quad (1.3)$$

де R, G, B – нормовані компоненти кольору у моделі RGB.

У разі використання ахроматичних кольорів можна вважати, що $R = G = B$, отже формула (1.3) перетворюється на рівність $L = L_{halftone}$. Тоді для випадку, коли використовуються білий, чорний та деякий напівтоновий колір, цей колір визначатиметься як вектор нормованих колірних координат $(0.179, 0.179, 0.179)$, що відповідає відтінку сірого кольору середньої інтенсивності. Таким чином, цілком виправданим є розроблення чорно-сіро-білого коду для кодування інформації про об'єкти логістичного обліку.

Застосування трьох кольорів або відтінків кольору з точки зору кодування даних означає перехід до трійкової системи числення. При цьому, обсяг даних, що можуть бути подані у вигляді ШК-позначки, зростає.

Запропонований підхід до вибору кількості кольорів дозволяє забезпечити високу інформаційну ємність ШК-позначки та максимальну контрастність штрихкодового зображення, що у свою чергу знижуватиме ризик виникнення помилки розпізнавання колірної інформації при декодуванні штрихкодованих даних.

1.3.2. Особливості виникнення спотворень при використанні ШК-позначок

Внесення спотворень в ШК-елементи можливе на етапі виготовлення ШК-позначки або під час її зберігання та переміщення.

Причинами спотворень в ШК-елементах під час виготовлення ШК-позначки може бути наступне.

1. Дефекти носія – нерівність поверхні, низька якість матеріалу, неправильний вибір матеріалу тощо.
2. Низька якість використовуваного барвника (фарби) або особливості зчеплення барвника з носієм – розпливання або згортання фарби, неконтрастне зображення тощо.
3. Низька якість та дефекти друку внаслідок збоїв друкувального пристрою.

Під час зберігання та переміщення ШК-позначок причинами спотворень ШК-елементів може бути наступне.

1. Механічні ушкодження або деформації ШК-елементів (наприклад, під час транспортування чи переміщення об’єкта із ШК-позначкою).
2. Забруднення частини ШК-позначки або поява неконтрастності через старіння носія та барвника.

Таким чином, у подальшому досліджені розрізнятимемо 6 видів ушкоджень ШК-елементів (табл. 1.2), які переважно виникають в процесі друку або переміщення ШК-позначки.

Перший вид ушкоджень (поява одного чи декількох додаткових ШК-елементів) може бути зумовлений забрудненням (наприклад, подряпиною) ШК-позначки.

Таблиця 1.2. Можливі ушкодження в ШК-знаках

Вид ушкодження	Причина ушкодження	Вид помилки
1. Поява додаткового (-их) ШК-елемента (-ів)	Змішування двох кольорів	Однократна (багатократна) помилка многозначного символу
	Забруднення, подряпина	
2. Випадання ШК-елемента	Розпливання фарби	Однократна (багатократна) помилка многозначного символу
3. Зменшення розмірів ШК-елемента	Деформація (вм’ятина) носія	Помилка випадання Многозначного символу
4. Збільшення розмірів ШК-елемента	Деформація (розрив) носія	Помилка вставки многозначного символу
5. Взаємне зміщення ШК-елементів	Згортання (розтікання) фарби	Однократна (багатократна) помилка многозначного символу
6. Зміна кольору ШК-елемента	Змішування кольорів, забруднення	Однократна (багатократна) помилка многозначного символу

Другий вид ушкоджень (випадання ШК-елемента) може бути спричинений, наприклад, розпливанням фарби під час друкування одного із ШК-елементів. Це може призвести до зникнення одного елемента розміром в 1 мод^2. Такий вид ушкоджень призводить до появи помилки многозначного символу. Якщо має місце випадання ШК-елементів завширшки понад 1 мод, то помилка буде пакетною.

Третій вид ушкоджень (зменшення ШК-елемента) зумовлено деформацією (наприклад, вм’ятиною) носія і призводить до помилки випадання многозначного символу.

Четвертий вид ушкоджень (розрив носія) збільшує один із ШК-елементів, в результаті чого можлива поява помилки вставки многозначного символу.

П’ятий вид ушкоджень (взаємне зміщення ШК-елементів) може бути спричинений згортанням або розтіканням фарби та призводить до однократної (у загальному випадку – пакетної) помилки многозначного символу.

Шостий вид ушкоджень може спричинюватися змішуванням кольорів або забрудненням носія і призводить до зміни кольору ШК-елемента. При цьому кількість ШК-елементів у ШК-знаку не змінюється. Наслідком такого ушкодження може бути поява помилки многозначного символу.

Коли ШК-позначка наноситься гравіюванням (наприклад, на металеву поверхню), то найімовірнішими є помилки першого (подряпина), третього та четвертого типів.

Для забезпечення можливості відновлення даних з пошкодженої ШК-позначки доцільно застосовувати алгоритмічно-програмні методи забезпечення завадостійкості.

Отже, при формуванні вимог до програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів потрібно визначити вимогу щодо необхідності включення до складу програмної системи компонента, який забезпечуватиме завадостійке кодування та декодування даних.

1.4. Висновки до першого розділу

Проведене дослідження методів автоматичної ідентифікації об’єктів логістики на основі технології штрихового кодування та особливостей застосування штрихових кодів дозволяє зробити наступні висновки.

1. Доцільно розширити спектр застосування елементів технології штрихового кодування в програмних системах у галузі логістики.
2. Існуючі монохромні штрихові коди дозволяють подавати у вигляді ШК-позначок обмежені обсяги інформації та не надають можливості адаптації вхідного алфавіту до потреб користувачів.
3. Існуючі кольорові двовимірні штрихові коди мають високу інформаційну ємність, проте використання багатоколірності призводить до підвищення обчислювальної складності алгоритму декодування ШК-позначок та посилення вимог до апаратного забезпечення процесів виготовлення та розпізнавання ШК-позначок.
4. Доцільно зменшити кількість використовуваних у кольорових штрихових кодах кольорів до трьох, що спонукає до розроблення нових алгоритмічно-програмних методів, які ґрунтуватимуться на використанні трійкової системи числення, для формування даних на основі високощільних штрихових кодів з трьома градаціями кольору.
5. Для підвищення завадостійкості ШК-позначок з трьома градаціями кольору доцільно розробити новий алгоритмічно-програмний метод забезпечення завадостійкості штрихових кодів, який ґрунтуватиметься на використанні трійкової системи числення.

РОЗДІЛ 2. АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПРОЦЕСІВ ФОРМУВАННЯ ДАНИХ НА ОСНОВІ ШТРИХОВИХ КОДІВ З ТРЬОМА ГРАДАЦІЯМИ КОЛЬОРУ

2.1. Задача забезпечення компактного подання даних у вигляді штрихкодових зображень

Штрихкодове подання даних пов’язане не лише з перетворенням інформації з однієї форми (текстової) на іншу (машиночитану), але й з необхідністю фізичного розміщення штрихкодової позначки на об’єкті, що підлягає ідентифікації. Припустимі фізичні розміри ШК-позначки визначаються не тільки об’ємом інформації, яку потрібно подати, а ще й залежать від максимально можливої площі носія ШК-позначки та технології нанесення штрихового коду (друк, лазерне гравіювання тощо). Тому задача забезпечення компактного подання даних у вигляді штрихкодових зображень, в тому числі шляхом ущільнення даних, є надзвичайно актуальною.

У першому розділі дисертації було визначено, що доцільно застосовувати штрихове кодування даних на основі штрихових кодів з трьома градаціями кольору. Тому задача, що вирішується у цьому розділі, полягає у розробленні алгоритмічно-програмних підходів до забезпечення компактного подання даних у вигляді штрихових кодів з трьома градаціями кольору завдяки:

- ущільнення даних при їх підготовці до подання у вигляді штрихового коду;
- структурне ущільнення штрихкодованих даних шляхом їх подання у вигляді дворівневого штрихового коду.

Ці два підходи можуть бути, залежно від зовнішніх умов, як-от: тип даних, розмір носія, потреба у розділені доступу до інформації тощо – застосовані окремо один від одного або об’єднані.

2.2. Структурний метод ущільнення даних при формуванні штрихкодових позначок

Двовимірною штрихкодовою позначкою (ДШК-позначкою) називатимемо сукупність штрихкодових знаків, що компактно розташовані на носії у вигляді матриці. *Штрихкодовий знак* (ШК-знак) – це графічне представлення *s*-розрядної трійкової послідовності символів. Відповідний *s*-розрядний трійковий код, що відповідає ШК-знаку, називатимемо кодовим вектором.

Залежно від розрядності ШК-знак має різну форму, варіації якої для кожного s наведено на рис. 2.1.

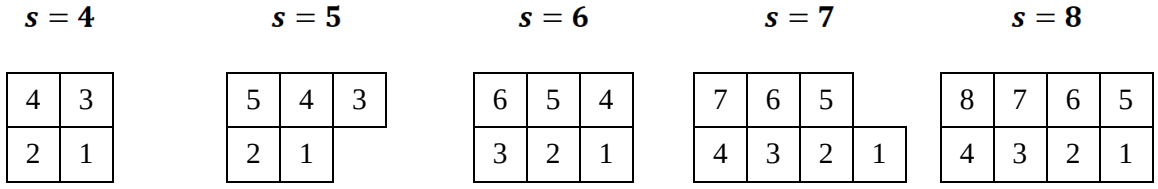


Рис. 2.1. Порядок розміщення бітів у ШК-знаку залежно від його розрядності

Розрізнятимемо два види ШК-знаків: інформаційні та службові.

Інформаційні ШК-знаки використовуються для подання у штриховому вигляді вхідної алфавітно-цифрової послідовності символів. Службові ШК-знаки використовуються для переходів між режимами кодування даних, команд сканеру, символів заповнення замість ШК-знаків,

яких не вистачає для формування завершеної фігури ДШК-позначки, контрольних символів при завадостійкому кодуванні тощо.

ШК-знак складається з s елементів, які у фізичному сенсі являють собою комірки матриці. Кожній з комірок ставиться у відповідність цифровий еквівалент однієї з трьох градацій обраного для використання кольору. В цьому дисертаційному дослідженні пропонується розглядати три градації сірого кольору, які умовно назвемо як білий, якому відповідатиме цифровий еквівалент 0, сірий, якому відповідатиме цифровий еквівалент 1, та чорний, якому відповідатиме цифровий еквівалент 2.

Тоді максимальна ємність ДШК-позначки (тобто потужність множини всіх можливих ШК-знаків) складатиме $V_{max} = 3^s$ ШК-знаків. У табл. 2.1 наведена відповідність значення параметру s, який являє собою розрядність ШК-знаку, максимальній ємності ДШК-позначки.

Таблиця 2.1. Залежність максимальної ємності ДШК-позначки від розрядності ШК-знаку

s	3^s	V_{max}
4	3^4	81
5	3^5	243
6	3^6	729
7	3^7	2187
8	3^8	6561
9	3^9	19683
10	3^{10}	59049

Розглядати ДШК-позначки з параметром $s < 4$ є беззмістовним, оскільки відповідні ДШК-позначки матимуть надзвичайно малу ємність, непридатну для практичного застосування. Тому в цьому дослідженні

розглядатимуться розрядності ШК-знаку $s = 4, \ldots, 10$, які становлять практичний інтерес та відповідають ДШК-позначкам ємністю від 81 до 59049 ШК-знаків.

Множину всіх можливих ШК-знаків за фіксованого s називатимемо *символікою* Ω *штрихового коду*. При цьому, вважатимемо цю символіку алфавітом Ω потужності $P_\Omega = 3^s$. Цей алфавіт складається з інформаційних Ω_{inf} та службових Ω_{aux} ШК-знаків, тобто $\Omega = \Omega_{inf} \cup \Omega_{aux}$. Потужність алфавіту інформаційних ШК-знаків дорівнює $P_{\Omega_{inf}}$, службових – $P_{\Omega_{aux}}$. Таким чином, $P_{\Omega_{inf}} + P_{\Omega_{aux}} = 3^s$. Для подання інформації у штрихкодовому вигляді використовуватимемо $P_{\Omega_{inf}}$ ШК-знаків, тоді як службовими є $P_{\Omega_{aux}}$ ШК-знаків символіки.

Таким чином, алфавіт Ω символіки ДШК складається з множини ШК-знаків. Кожному ШК-знаку, що входить до складу ДШК, відповідає кількісний еквівалент – порядковий номер ШК-знаку в символіці Ω, тобто символіку Ω можна розглядати як числову множину $\Omega = \{0, 1, 2, \ldots, P_\Omega - 1\}$, де P_Ω – потужність множини (символіки) Ω; при цьому $P_\Omega = 3^s$, де s – кількість елементів, що входять до складу кожного ШК-знаку.

Вхідна інформація, яка в подальшому буде перетворена на символіку Ω , являє собою послідовність символів, які належать до алфавіту A. Алфавіт A складається з літер латинського та кириличного алфавіту, цифр і спеціальних знаків та є підмножиною алфавіту ASCII, тобто $A \subset \text{ASCII}$.

Аналогічно алфавіту символіки Ω, кожному символу з алфавіту A відповідає кількісний еквівалент – порядковий номер символу в алфавіті A, тобто алфавіт A можна представити як числову множину $A = \{0, 1, 2, \ldots, P_A - 1\}$, де P_A – потужність множини (алфавіту) A.

В подальшому будемо оперувати двома множинами:

– алфавіт A потужності P_A;

– алфавіт Ω потужності P_Ω.

При цьому, говоритимемо, що алфавіту A (числовій множині A) відповідає система числення з основою P_A, а, відповідно, алфавіту Ω_{inf} (числовій множині Ω_{inf}) – система числення з основою $P_{\Omega_{inf}}$. Таким чином, перетворення утворених з символів алфавіту A текстових даних на символи алфавіту Ω_{inf} (тобто перетворення алфавітно-цифрових даних у штрихкодову форму) реалізовуватимемо як перетворення числа з системи числення з основою P_A у систему числення з основою $P_{\Omega_{inf}}$.

У вигляді ДШК-позначки може бути подана будь-яка інформація, представлена у вигляді алфавітно-цифрової послідовності символів.

При поданні даних у вигляді ДШК-позначки вирішуються дві задачі:

1) ущільнення алфавітно-цифрової послідовності даних, які потрібно подати у штрихкодовому вигляді;
2) забезпечення завадостійкості штрихкодової позначки.

Розглянемо першу задачу – ущільнення алфавітно-цифрової послідовності символів для її подальшого подання у вигляді штрихового коду.

2.2.1. Ущільнення вхідної послідовності

Нехай маємо вхідну алфавітно-цифрову послідовність вигляду:

$$\mathbb{T} = t_1 t_2 \ldots t_h, t_i \in \text{ASCII}, i = 1, 2, \ldots, h \qquad (2.1)$$

де t_i – це елемент вхідної послідовності,

h – довжина вхідної послідовності $\mathbb{T}$,

ASCII – множина символів з алфавіту ASCII, які необхідно подати у штрихкодовому вигляді.

Зазначимо, що множина $\text{ASCII} = L \cup D \cup C$, де L – множина літер, D – множина цифр, C – множина спеціальних символів.

Послідовність $\mathbb{T}$ розбивається на суміжні підпослідовності, що містять елементи, які належать до однієї з множин:

$$\begin{cases} w_1 = t_1 t_2 \dots t_{i-1} \in L \\ w_2 = t_i t_{i+1} \dots t_{h-i+1} \in D \\ \dots \\ w_k = t_{h-i} t_{h-i-1} \dots t_h \in C \end{cases} \quad (2.2)$$

Таким чином, вхідна послідовність набуває вигляду:

$$\mathbb{T} = w_1 w_2 \dots w_k, \quad (2.3)$$

де w_i – це підпослідовність символів вхідної послідовності, яка містить елементи лише однієї з множин. При цьому, у вхідній послідовності $\mathbb{T}$ підпослідовності $w_1, w_2, \dots, w_k$ можуть слідувати у будь-якому порядку.

Розглянемо математичну модель перетворення вхідної послідовності $\mathbb{T}$ на ущільнену послідовність $\mathbb{U}$.

Підпослідовність $w_i = t_1 t_2 \dots t_n$ розглядається як n-розрядний вектор у P_A-ковій системі числення. В результаті ущільнення ця підпослідовність перетворюється на m-розрядний вектор у системі числення $P_{\Omega_{inf}}$, де $n > m$ та $P_A \leq P_{\Omega_{inf}}$. В результаті отримуємо послідовність j векторів $u_i = c_1 c_2 \dots c_m$ вигляду:

$$\mathbb{U} = u_1 u_2 \ldots u_j \,, \tag{2.4}$$

де u_i – підпослідовність довжиною j символів з ущільненої m-розрядної послідовності.

Процес ущільнення вхідної послідовності n суміжних символів, які належать до алфавіту A потужністю P_A, зводиться до їх перетворення у m ШК-знаків – символів алфавіту Ω_{inf} (символіки ДШК) потужності $P_{\Omega_{inf}}$, тобто:

$$n(P_A) \rightarrow m(P_{\Omega_{inf}}) \tag{2.5}$$

де n – кількість символів алфавіту A,

m – кількість символів алфавіту Ω_{inf}.

Перетворення (2.5) фактично являє собою процедуру перетворення числа з однієї системи числення до іншої системи числення. Тоді, n-розрядне число у системі числення з основою P_A перетворюється на відповідне число розрядністю m у системі числення з основою $P_{\Omega_{inf}}$, тобто:

$$\sum_{i=0}^{n-1} \alpha_i P_A^i \rightarrow \sum_{j=0}^{m-1} \omega_j P_{\Omega_{inf}}^j \,, \tag{2.6}$$

де α_i – відповідні коди символів з алфавіту A,

ω_i – відповідні коди символів у алфавіті Ω_{inf}.

Перетворення (2.5) відбуватиметься з ущільненням, якщо $n \log_3 P_A > m \left\lfloor \log_3 P_{\Omega_{inf}} \right\rfloor$, але при цьому має виконуватись співвідношення

$P_A^n - 1 \le P_{\Omega_{inf}}^m - 1$, де $P_A^n - 1$ – це кількісний еквівалент максимального n-розрядного числа в системі числення з основою P_A, а $P_{\Omega_{inf}}^m - 1$ – кількісний еквівалент максимального m-розрядного числа в системі числення з основою $P_{\Omega_{inf}}$.

Таким чином, щоб вхідна алфавітно-цифрова послідовність $\mathbb{T}$ завдовжки n символів з алфавіту A була ущільнена у послідовність ШК-знаків $\mathbb{U}$ завдовжки m символів із символіки Ω, яка в результаті й буде представлена у вигляді ДШК-позначки, необхідно та достатньо, щоб виконувалась умова:

$$\begin{cases} P_A^n - 1 \le P_{\Omega_{inf}}^m - 1 \\ n \log_3 P_A > m \left\lfloor \log_3 P_{\Omega_{inf}} \right\rfloor \end{cases}, \tag{2.7}$$

що зводиться до вигляду:

$$\begin{cases} P_A^n \le P_{\Omega_{inf}}^m \\ n \log_3 P_A > m \left\lfloor \log_3 P_{\Omega_{inf}} \right\rfloor \end{cases}, \tag{2.8}$$

де $n \log_3 P_A$ – довжина вхідної неущільненої послідовності символів,

$m \left\lfloor \log_3 P_{\Omega_{inf}} \right\rfloor$ – довжина ущільненої послідовності.

Тоді задача полягає у знаходженні такого P_A при фіксованому $P_{\Omega_{inf}}$, щоб забезпечувалось максимальне ущільнення повідомлення при його перетворенні на ДШК-позначку.

Ступінь ущільнення вхідних даних визначатимемо як співвідношення довжини вхідної послідовності та довжини ущільненої послідовності та називатимемо *коефіцієнтом ущільнення*:

$$U^{(s)}_{P_{\Omega_{inf}}}(P_A) = \frac{n \log_3 P_A}{m \left\lceil \log_3 P_{\Omega_{inf}} \right\rceil} \tag{2.9}$$

З точки зору кінцевого результату, ущільнена вхідна послідовність, порівняно з неущільненою, буде представлена ДШК-позначкою меншої площі, що забезпечує більш гнучке та компактне розміщення її на об'єкті-носії. Крім цього, ущільнення даних надає можливість збільшувати інформаційну ємність ДШК-позначки за її незмінної площі, що може використовуватись для збільшення обсягів закодованої інформації.

Для обчислення коефіцієнтів ущільнення розв'яжемо систему нерівностей (2.7). В результаті отримуємо множину цілочислових розв'язків. Всі розв'язки цієї системи для $s = 4, 6, 8$ наведено у Додатку Г.

У лістингу 2.1 мовою C# наведено програмний метод отримання всіх цілочислових розв'язків системи.

Лістинг 2.1. Програмний метод отримання розв'язків системи нерівностей

```
private List<double[]> GetSolutions(int digitCapacity, int pOmegaAux)
{
    int pOmega = Convert.ToInt32(Math.Pow(3, digitCapacity));
    int pOmegaInf = pOmega - pOmegaAux;
    double log3POmegaInf = Math.Ceiling(Math.Log(pOmegaInf, 3));
    double log3pA, compressionCoefficient;
    List<double[]> resultingTableList = new List<double[]>();

    for (int pA = 10; pA <= 256; pA++)
    {
        for (int n = 1; n <= 20; n++)
        {
            for (int m = 1; m <= 10; m++)
            {
                log3pA = Math.Ceiling(Math.Log(pA, 3));

                if ((Math.Pow(pA, n) <= Math.Pow(pOmegaInf, m)) &&
                            (n * log3pA > m * log3POmegaInf))
                {
                    compressionCoefficient = (n * log3pA) / (m * log3POmegaInf);

                    if ((compressionCoefficient > 1) && (m < n))
                        resultingTableList.Add(new double[4]
                                { n, m, pA, compressionCoefficient });              }
            }        }    }

    return resultingTableList;     }
```

Варто зауважити, що у розробленому програмному забезпеченні робиться обмеження на потужність алфавітів 256 та довжину вхідної підпослідовності 20. Це зроблено з причини того, що, по-перше, немає сенсу розглядати алфавіти, потужність яких більша за потужність розширеного алфавіту ASCII, особливо враховуючи те, що зі збільшенням потужності зменшується коефіцієнт ущільнення. По-друге, здебільшого ефективніше кодувати короткі підпослідовності, оскільки це дозволить переходити між алфавітами у разі, якщо зустрінеться підпослідовність, всі символи якої належать іншому алфавіту з більшим ущільненням.

З множини отриманих розв'язків ще на етапі її формування відкидаються розв'язки, для яких коефіцієнт ущільнення дорівнює одиниці, оскільки це означає що символи вхідної послідовності будуть перетворені без ущільнення, що не становить інтересу для цього дослідження. Також можуть бути відкинуті розв'язки з ущільненням, меншим за 10%, оскільки практична цінність такого ущільнення, особливо на маленьких обсягах даних, є низькою.

Втім, залишається забагато можливих варіантів, з яких потрібно обрати доцільні для практичного використання алфавіти. Процедура обрання може бути здійснена людиною – експертом у предметній галузі, залученим до розроблення певної програмної системи, однак є неефективним рішенням, оскільки зі збільшенням розрядності ШК-знаку збільшуватиметься й кількість розв'язків системи (2.7). Відповідно, доцільнішим є використання багатокритеріальної оптимізації, яка дозволила б значно скоротити множину розв'язків із подальшим обранням експертом або користувачем програмної системи, що реалізує створення штрихкодової позначки, релевантних для конкретної галузі алфавітів.

2.2.2. Багатокритеріальна оптимізація для визначення потужності алфавіту

Для скорочення множини допустимих розв'язків пропонується використовувати багатокритеріальну оптимізацію. Критеріями у цьому випадку виступатимуть складові розв'язку системи (2.7), а саме: n – кількість символів початкової послідовності, m – кількість символів послідовності після ущільнення, P_A – потужність алфавіту початкової послідовності, d – тип перетворення $n \to m$, де $d = n - m$, та $U^{(s)}_{P_{\Omega_{inf}}}$ – коефіцієнт ущільнення вхідної послідовності.

Залежно від конкретного практичного випадку метою оптимізації може бути максимізація одних критеріїв та мінімізація інших. Умови багатокритеріальної оптимізації визначає експерт або користувач певної програмної системи.

Існують різні підходи [14] до багатокритеріальної оптимізації, які можна класифікувати наступним чином: методи генерування недомінуючих рішень (вагові та обмежувальні методи); методи, що базуються на обчисленні відстаней (наприклад, компромісне програмування, цільове програмування, методи опорних точок); та інтерактивні методи.

Для вирішення задачі багатокритеріальної оптимізації для визначення потужності алфавіту в цьому дослідженні використовуватиметься підхід Парето, який дозволяє звузити множину всіх розв'язків та отримати множину парето-оптимальних розв'язків. Цей підхід полягає у попарному порівнянні розв'язків початкової множини, за якого мають виконуватись аксіома Парето та аксіома виключення. Сформулюємо їх.

Аксіома Парето полягає у тому, що якщо оцінка одного з двох варіантів не гірше оцінки другого варіанту за всіма критеріями, причому

принаймні один з них – строго краще, то перевага надається першому варіанту, тобто:

$$x', x'' \in X,\ f_i(x') \geq f_i(x''), i = 1,\ldots,m, \qquad (2.10)$$
$$\exists k \in \{1,\ldots,m\}: f_k(x') > f_k(x'') \Rightarrow x' \succ_X x'',$$

де x', x'' – пара розв'язків, які порівнюються, з множини всіх розв'язків X;

$f_i(x)$ – цільова функція критерію для розв'язку x;

m – кількість критеріїв;

$\succ_X$ – бінарне відношення переваги розв'язку ліворуч від знаку над розв'язком праворуч.

Аксіома виключення каже про те, що варіант, не обраний у будь-якій парі розв'язків, не повинен потрапити у множину обраних розв'язків та має бути виключений з початкової множини можливих варіантів, тобто:

$$x', x'' \in X, x' \succ_X x'' \Rightarrow x'' \notin C(X), \qquad (2.11)$$

де $C(X)$ – множина обраних розв'язків.

Ці всі аксіоми уможливлюють фундаментальний принцип багатокритеріального вибору Еджворта-Парето: якщо аксіома Парето та аксіома виключення виконуються для будь-якої множини обраних розв'язків $C(X)$, то має місце включення $C(X) \subset P_f(X)$, де $P_f(X)$ – множина парето-оптимальних варіантів, яка визначається наступним чином:

$$P_f(X) = \{x^* \in X \mid \textit{не існує } x \in X \textit{ такого, що } f_i(x) \geq f_i(x^*),$$

$$i = 1,\ldots,m,\ f(x) \neq f(x^*)\}.$$

Тоді алгоритм побудови множини парето-оптимальних розв'язків складається з семи кроків, наведених на рис. 2.2.

Отже, використовуватимемо багатокритеріальну оптимізацію за підходом Парето для визначення оптимальних алфавітів, які забезпечать найкращу ємність ШК-позначки.

Кожний розв'язок, отриманий зі системи нерівностей (2.7), може бути представлений у вигляді вектору, що складається з відповідних складових розв'язку n , m , P_A , d та $U^{(s)}_{P_{\Omega_{inf}}}$. Відповідно, множину розв'язків можна розглядати як множину векторів, кожний елемент яких є відповідним критерієм для оптимізації. В загальному випадку можна говорити про п'ять критеріїв, за кожним з яких має бути здійснена максимізація або мінімізація.

Втім, на практиці така оптимізація беззмістовна, оскільки деякі з цих критеріїв є взаємозалежними: йдеться про параметри n , m та d . Варто взяти до уваги також той факт, що серед розв'язків системи (2.7) зазвичай є ціла низка розв'язків, які відрізняються виключно потужністю алфавіту, що ускладнює оптимізацію за усіма параметрами. Тому замість цього доцільніше використовувати «налаштовану» оптимізацію, оптимізуючи, по-перше, за тими параметрами, які є найпринциповішими, а по-друге, в межах обраних розв'язків, які представляють найбільший інтерес.

Розглянемо це на прикладі розв'язків для $s = 4$. У табл. 2.1 наведено фрагмент табл. Г.1 Додатку Г, який демонструє групи розв'язків, що відрізняються лише одним чи двома параметрами.

У цьому фрагменті можна побачити дві великі групи, об'єднані спільним параметром n (групи розв'язків $n = 5$ та $n = 7$). Якщо відомо наперед, що дані, які передбачається кодувати у програмній системі,

підпорядковуються певному шаблону (скажімо, код об’єкту логістики завжди складається з семи цифр), то межі розв’язків, які потребують оптимізації можна звузити до тих розв’язків, для яких $n = 7$. Тоді оптимізація відбуватиметься серед чотирьох розв’язків цього фрагменту.

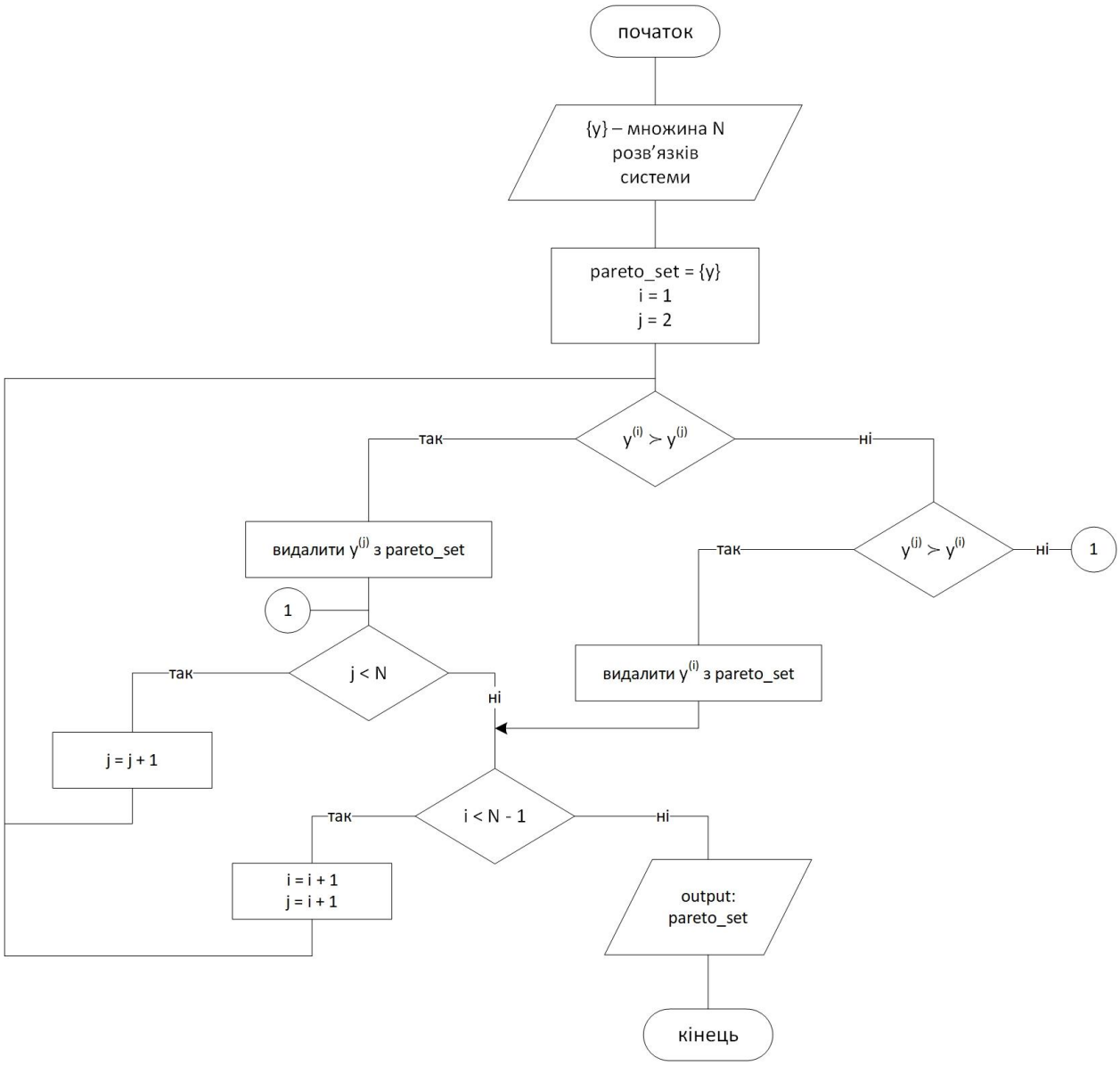


Рис. 2.2. Алгоритм побудови множини парето-оптимальних розв’язків

Визначивши ці межі, можна тепер визначити параметри, які стануть критеріями у підході Парето. Оскільки ми визначили, що код об’єкту логістики може містити лише цифри, за критерієм потужності алфавіту

доцільно шукати мінімум, тоді як за критеріями типу перетворення та коефіцієнту ущільнення нас цікавитиме максимізація, адже ми хотіли б отримати якомога більший відсоток ущільнення та такий тип перетворення, щоб сім символів коду переходили в якомога меншу кількість ШК-знаків.

Таблиця 2.1. Деякі цілочислові розв'язки системи (2.7) за $s = 4$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$ (різниця d)	Коефіцієнт ущільнення $U_{70}^{(4)}(P_A)$
4	5	29	$5 \to 4$ (1)	1,250
4	5	28	$5 \to 4$ (1)	1,250
3	5	12	$5 \to 3$ (2)	1,250
3	5	11	$5 \to 3$ (2)	1,250
4	7	11	$7 \to 4$ (3)	1,313
4	7	10	$7 \to 4$ (3)	1,313
6	7	38	$7 \to 6$ (1)	1,167
6	7	37	$7 \to 6$ (1)	1,167

Застосувавши алгоритм, наведений на рис. 2.2, для фрагменту, що розглядається, отримаємо два парето-оптимальних розв'язки (табл. 2.2).

Таблиця 2.2. Множина парето-оптимальних розв'язків для $n = 7$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$ (d)	Коефіцієнт ущільнення $U_{70}^{(4)}(P_A)$
4	7	10	$7 \to 4$ (3)	1,313
6	7	37	$7 \to 6$ (1)	1,167

Це одразу викриває недолік підходу Парето – здебільшого множина парето-оптимальних розв'язків складається з двох чи більше розв'язків,

що унеможливлює автоматичне отримання найоптимальнішого для конкретного випадку алфавіту. Хоча в деяких ситуаціях ефективнішим може бути аналіз отриманої множини експертом, який зможе проаналізувати сукупність факторів, та для наведеного вище прикладу обрати алфавіт потужністю 10, однак у загальному випадку доцільніше автоматично звужувати цю множину для отримання користувачем певної програмної системи однозначної відповіді.

Для звуження множини скористаємось методом лінійної згортки, який полягає у призначенні невід'ємних коефіцієнтів $\mu_1,\ldots,\mu_m$, які у сумі зазвичай мають складати одиницю, кожному з критеріїв із подальшою максимізацією лінійної комбінації критеріїв $\sum_{i=1}^{m}\mu_i f_i(x)$ на множині парето-оптимальних розв'язків. Функцію $\sum_{i=1}^{m}\mu_i f_i(x)$, яка має задовольняти умові:

$$x \succ x' \Leftrightarrow \sum_{i=1}^{m}\mu_i f_i(x) > \sum_{i=1}^{m}\mu_i f_i(x'), \qquad (2.12)$$

називають *лінійною функцією корисності*.

Коефіцієнти функції корисності визначаються експертом або користувачем програмної системи відповідно до конкретної предметної галузі. У випадку прикладу, що розглядався вище, найбільше нас цікавлять критерії потужності алфавіту (мінімізація), типу перетворення (максимізація) та коефіцієнту ущільнення (максимізація), для чого введемо відповідні коефіцієнти $\alpha = 0{,}3$; $\beta = 0{,}4$; $\gamma = 0{,}3$. Результатом застосування лінійної згортки є отримання остаточного розв'язку, який наведено у табл. 2.3.

Таблиця 2.3. Оптимальний розв’язок для $n = 7$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$ (d)	Коефіцієнт ущільнення $U_{70}^{(4)}(P_A)$
4	7	10	$7 \to 4$ (3)	1,313

Отриманий розв’язок повністю відповідає вимогам, адже здатний перетворити сім символів цифрового коду на чотири ШК-знаки з ущільненням понад 30%. Потужність пропонованого алфавіту складає 10, що також задовольняє десяти цифрам від 0 до 9, з яких може складатись код.

Таким чином, можна сформулювати два підходи до багатокритеріальної оптимізації розв’язків системи нерівностей (2.7):

1) отримання множини парето-оптимальних розв’язків серед однієї групи чи декількох груп розв’язків зі спільним параметром із подальшим застосуванням лінійної згортки для визначення оптимального алфавіту;
2) отримання множини парето-оптимальних розв’язків серед всіх розв’язків за обмеженою кількістю критеріїв із подальшим експертним аналізом визначеної множини відповідно до конкретної сфери застосування з метою виділення двох чи більше оптимальних алфавітів. Найпильніша увага зазвичай приділяється тим розв’язкам, які мають найбільший коефіцієнт ущільнення. Значення потужності алфавіту A у таких розв’язків аналізується з точки зору практичної застосовності.

Крім того, в деяких випадках доцільним може також стати поетапне застосування підходу Парето для попереднього звуження великої множини допустимих розв’язків із подальшою участю експерта для обрання груп розв’язків, в яких варто продовжити оптимізацію до

отримання достатньо невеликої множини парето-оптимальних розв'язків, з яких експерт зможе ефективно обрати релевантний алфавіт.

Багатокритеріальна оптимізація здебільшого використовується на етапі проєктування програмної системи для певної визначеної галузі, в якій наперед відомо, якими алфавітами послуговуватимуться користувачі цієї системи, а отже – остаточний вибір алфавітів може бути покладений на експерта у цій предметній галузі, який зможе оцінити всі варіанти та обрати найоптимальніший для конкретного випадку.

Якщо ж розглядати багатокритеріальну оптимізацію як складову системи формування штрихкодових позначок, то доцільніше максимально автоматизувати процес пошуку оптимальних розв'язків для можливості отримання єдиного рішення. Втім, зауважимо, що така система матиме певну похибку, в результаті чого пропонований користувачу варіант може бути не найоптимальнішим. Для уникнення цього доцільним є або використання додаткових методів звуження парето-оптимальної множини, або передбачення можливості ручного корегування остаточного розв'язку користувачем на основі отриманих парето-оптимальних розв'язків.

Повна програмна реалізація формування множини розв'язків системи (2.7) наведена мовою C# у Додатку Ґ.

Аналіз розв'язків для деяких s та дослідження процедури ущільнення вхідних даних наведені у п. 2.3.

2.2.3. Режими кодування даних

Як було зазначено у п. 2.2.1, в загальному випадку символи вхідної послідовності $\mathbb{T}$ належать до алфавіту ASCII. Потужність цього алфавіту складає $P_{\text{ASCII}} = 128$ та не включає в себе символи розширеного ASCII.

Згідно з (2.2) та (2.3) вхідна послідовність може бути розбита на множину підпослідовностей, символи кожної з яких належать або до

множини літер, або до множини цифр, або до множини спеціальних символів (тобто символів, які не є літерами та цифрами). Зокрема, серед цих підпослідовностей можуть зустрічатись довгі послідовності десяткових цифр або довгі послідовності літер. Для отримання більшого ущільнення необхідно мати можливість переходити між переліченими алфавітами. Для цього введемо поняття режиму кодування, яким будемо послуговуватись надалі.

Під *режимом кодування* розумітимемо алфавіт потужності P_k, де k – це один з визначених вище алфавітів, до якого належать всі суміжні символи підпослідовності $w_i \in \mathbb{T}$. Для переходу між алфавітами, а отже – режимами кодування, використовуватимемо відповідні службові символи S – перемикачі режимів.

Визначимо можливі режими кодування. Вхідну послідовність $\mathbb{T}$ можна розбити на підпослідовності суміжних символів, які утворюються чотирма алфавітами:

– базовий алфавіт A, ASCII (підпослідовності T_A);

– алфавіт десяткових цифр D_{10} (підпослідовності $T_{D_{10}}$);

– алфавіт шістнадцяткових цифр D_{16} (підпослідовності $T_{D_{16}}$);

– алфавіт текстових символів L (підпослідовності T_L).

Залежно від розрядності s, яка розглядається, а відповідно – від коефіцієнту ущільнення, алфавіт десяткових цифр може бути зведений до алфавіту шістнадцяткових цифр, а отже – підпослідовності $T_{D_{10}}$ зводяться до підпослідовностей $T_{D_{16}}$. Втім, в загальному випадку розглядатимемо у тому числі й десятковий режим.

Таким чином, в загальному випадку кодування символів вхідної послідовності може відбуватись шляхом перемикання між трьома основними режимами:

– режим A (ASCII режим) – для підпослідовностей символів з алфавіту ASCII;

– режим L (текстовий режим) – для підпослідовностей символів з алфавіту L;

– режим D_{10} (десятковий режим) – для підпослідовностей символів з алфавіту D_{10};

– режим D_{16} (шістнадцятковий режим) – для підпослідовностей символів з алфавіту D_{16}.

Перелічені режими вважатимемо мінімальним набором режимів, необхідним для досягнення максимального ущільнення вхідних даних. Втім, залежно від s ці режими можуть змінюватись – кількість режимів та їхній тип залежить від кількості алфавітів, визначених для конкретного s у конкретній предметній галузі.

Таким чином, вхідна послідовність $\mathbb{T}$ довжини h набуває наступного вигляду:

$$\mathbb{T}_h = T_{A,\alpha}\left[S_L \mid S_{D_{10}} \mid S_{D_{16}}\right]\ \left[T_{L,\beta} \mid T_{D_{10},\gamma} \mid T_{D_{16},\delta}\right] \qquad (2.13)$$

$$\left[S_L \mid S_{D_{10}} \mid S_A \mid S_{D_{16}}\right]\left[T_{A,\alpha} \mid T_{L,\beta} \mid T_{D_{10},\gamma} \mid T_{D_{16},\delta}\right]$$

де α – довжина підпослідовності з алфавіту A;

β – довжина підпослідовності з алфавіту L;

γ – довжина підпослідовності з алфавіту D_{10};

δ – довжина підпослідовності з алфавіту D_{16};

S_i – службовий символ-перемикач з поточного режиму в режим $i \in A, L, D_{10}, D_{16}$;

– повторення;

– необов’язковість використання.

У табл. 2.4 наведено схему можливих переходів між режимами ущільнення.

У разі введення більшої кількості режимів та/або відсутності одного чи більше наведених у таблиці режимів, множина S може бути розширена додатковими перемикачами, що, в свою чергу, потягне збільшення кількості службових ШК-знаків у алфавіті Ω_{aux}.

Таблиця 2.4. Переходи між режимами кодування алфавітно-цифрових послідовностей

Перехід в / Перехід з	Режим ASCII (A)	Десятковий режим (D_{10})	Шістнадцятковий режим (D_{16})	Текстовий режим (L)
Режим ASCII (A)	–	$S_{D_{10}}$	$S_{D_{16}}$	S_L
Десятковий режим (D_{10})	S_A	–	$S_{D_{16}}$	S_L
Шістнадцятковий режим (D_{16})	S_A	$S_{D_{10}}$	–	S_L
Текстовий режим (L)	S_A	$S_{D_{10}}$	$S_{D_{16}}$	–

Дослідження процедури обрання режимів даних залежно від розрядності ШК-знаку наведено у п. 2.3.

2.2.4. Підготовка вхідної послідовності символів до процедури кодування

Для застосування процедури кодування даних необхідно поділити вхідну послідовність на підпослідовності з урахуванням типів даних, наявних режимів кодування та можливого ступеня ущільнення даних.

Таким чином, з метою досягнення якнайвищого ступеня ущільнення даних процедура підготовка вхідної послідовності символів має забезпечувати:

- перевірку типу символів вхідної послідовності,
- розбиття вхідної послідовності на підпослідовності,
- додавання символів-перемикачів для позначення режиму кодування.

При цьому можливі два випадки:

1) набори даних, до яких належать вхідні символи, не перетинаються (наприклад, набір цифр, набір літер, набір знаків пунктуації);
2) набори даних перетинаються (наприклад, набір цифр, алфавітно-цифровий набір, набір символів ASCII).

Сформулюємо основні кроки алгоритму підготовки вхідної послідовності символів для першого випадка.

Крок 0. Отримання параметрів ущільнення $n^{(q)}$ та $m^{(q)}$ для кожного q-го набору даних.

Крок 1. Визначення належності кожного вхідного символу до певного набору символів.

Крок 2. Формування K груп символів одного типу, які послідовно розташовані у вхідній послідовності.

Крок 3. Визначення довжини p_k кожної групи.

Крок 4. Якщо $p_k \le n^{(q)}$ для k-ї групи символів, що належить q-му набору, то додати після k-ї групи символ-перемикач режиму для переходу до q-го набору.

Крок 5. Якщо $p_k > n^{(q)}$ для k-ї групи символів, що належить q-му набору, то поділити k-у групу на $t+1$ послідовних підгруп, де $t = p_k \bmod n^{(q)}$, причому t перших підгруп матимуть довжину $n^{(q)}$, а остання $t+1$-ша підгрупа матиме довжину $q = p_k - t \cdot n^{(q)}$, та додати після k-ї групи символ-перемикач режиму для переходу до q-го набору.

Кроки 4 та 5 потрібно виконати для кожної з K груп символів одного типу.

Сформулюємо основні кроки алгоритму підготовки вхідної послідовності символів для другого випадка.

Крок 0. Отримання параметрів ущільнення $n^{(q)}$ та $m^{(q)}$ для кожного q-го набору даних.

Крок 1. Визначення належності кожного вхідного символу до певного набору символів.

Крок 2. Формування K груп символів одного типу, які послідовно розташовані у вхідній послідовності, з позначенням подвійної належності відповідних символів.

Крок 3. Визначення довжини p_k кожної групи.

Крок 4. Якщо $p_k \le n^{(q)}$ для k-ї групи символів, що належить q-му набору, то додати після k-ї групи символ-перемикач режиму для переходу до q-го набору. Виконати крок для кожної з K груп символів одного типу.

Крок 5. Якщо $p_k > n^{(q)}$ для k-ї групи символів, що належить q-му набору, то поділити k-у групу на $t+1$ послідовних підгруп, де $t = p_k \bmod n^{(q)}$, причому t перших підгруп матимуть довжину $n^{(q)}$, а остання $t+1$-ша підгрупа матиме довжину $r = p_k - t \cdot n^{(q)}$. Зазначену дію виконати для кожного q-го набору, до якого належать символи, що мають подвійну належність. Виконати крок для кожної з K груп символів одного типу.

Крок 6. З можливих варіантів обрати той, за якого значення r буде меншим та додати після k-ї групи відповідний символ-перемикач режиму для переходу до q-го набору.

Програмний код, що реалізує процедуру підготовки вхідної послідовності символів, наведено у Додатку Д.

2.2.5. Етапи структурного методу ущільнення даних при їх поданні у вигляді штрихового коду з трьома градаціями кольору

Наведемо етапи, з яких складається структурний метод ущільнення даних при формуванні штрихкодових позначок з трьома градаціями кольору.

Етап 1. Розроблення системи штрихового кодування з ущільненням. Вхідними даними на цьому етапі є набір базових інформаційних одиниць, які фактично являють собою алфавіти конкретної предметної галузі, розрядність s ШК-знаку, символіка Ω ДШК-позначки та параметри для оптимізації. Цей етап виконується під час налаштування програмної системи та складається з наступних кроків.

Крок 1. Знаходження множини всіх можливих потужностей алфавітів у визначеній символіці Ω.

Крок 2. Для визначеної на попередньому кроці множини потужностей знаходження множини парето-оптимальних розв'язків та визначення оптимальних алфавітів відповідно до параметрів оптимізації.

Крок 3. Визначення на основі обраних оптимальних алфавітів A_i $(i = 1, 2, \ldots, k)$ режимів кодування, які використовуватимуться на Етапі 2.

Етап 2. Кодування вхідних даних з ущільненням. Вхідними даними на цьому етапі є алфавітно-цифрова послідовність. Цей етап виконується для кожної вхідної послідовності даних, використовуючи налаштування, визначені на Етапі 1.

Крок 1. Розбиття вхідної алфавітно-цифрової послідовності $t_1 t_2 \ldots t_h$, де h – довжина вхідної послідовності, на групи підпослідовностей $w_1 w_2 \ldots w_k$ кодів алфавітів A_i відповідно до визначених режимів кодування.

Крок 2. Перетворення груп кодів $w_1 w_2 \ldots w_k$ у символіку Ω_{inf} ДШК-позначки відповідної ємності: $n(A_i) \to m(P_{\Omega_{inf}})$.

Крок 3. Формування даних для ДШК-позначки шляхом перетворення отриманих на Кроці 2 кодів у трійкові кодовектори, які є цифровим еквівалентом ШК-знаку.

Таким чином, вхідні алфавітно-цифрові дані перетворюються на трійкові кодовектори, з яких й формується фінальна ДШК-позначка.

2.3. Дослідження процедури ущільнення даних

У п. 2.2 було наведено теоретичні засади процесів ущільнення даних при створенні штрихкодової позначки. Наведемо тепер дослідження цих процесів для деяких значень параметру розрядності ШК-знаку s.

Розглянемо випадок, коли $s = 4$. У такому разі $P_\Omega = 3^4 = 81$. Приймемо $P_{\Omega_{aux}} = 11$, відповідно $P_{\Omega_{inf}} = P_\Omega - P_{\Omega_{aux}} = 70$. Тоді, система нерівностей (2.7) матиме наступний вигляд:

$$\begin{cases} P_A^n \le 70^m \\ n \log_3 P_A > 4m \end{cases} \tag{2.14}$$

У табл. Г.4 наведено фрагмент таблиці Г.1, яка містить всі цілочислові розв’язки цієї системи.

На рис. Г.1 наведено залежність коефіцієнту ущільнення від потужності алфавіту. Варто зауважити, що на графіку коефіцієнт ущільнення дорівнює одиниці поблизу значення 3^s. Це відбувається тому, що службові символи виділені в окрему множину та не є частиною

алфавіту, а отже маємо нове значення $3^s - P_{\Omega_{aux}}$. Інакше, коефіцієнт ущільнення набував би значення одиниця точно у точці 3^s .

Вважатимемо, що нашою метою є створення системи із чотирма стандартними режимами, відповідно нас цікавлять алфавіти потужністю 10 (десятковий режим), 16 (шістнадцятковий режим) та деякий алфавіт, що дозволив би кодувати текстові послідовності. Нехай цей алфавіт в ідеальному випадку міститиме 37 символів – великі латинські літери та набір з 11 спеціальних символів. Режим ASCII розглядатись не буде, оскільки максимальна потужність за $s = 4$ дорівнює $P_A = 47$, що не покриває 128 символів базового алфавіту ASCII.

Тоді в наведеній множині розв'язків можемо виділити групи розв'язків, потужність алфавітів яких є максимально близькою (з обмеженням згори) до названих вище потужностей 10, 16 та 37. Такими потужностями є власне потужності P_{10} та P_{16}, які повністю відповідають десятковому та шістнадцятковому алфавітам, а також потужність P_{37} та найближча до неї потужність P_{38}.

Для визначення оптимального розв'язку для кожного з цих режимів маємо застосувати багатокритеріальну оптимізацію в межах трьох визначених груп: потужності P_{10}, потужності P_{16} та потужності P_{37} і P_{38}. Критеріями в усіх трьох групах визначимо коефіцієнт ущільнення (максимізація, параметр $\alpha = 0,4$), кількість символів початкової підпослідовності (мінімізація, параметр $\beta = 0,3$) та тип перетворення (максимізація, параметр $\gamma = 0,3$), оскільки нас цікавить можливість кодувати з максимальним ущільненням якомога коротші підпослідовності.

В результаті виконання процедури оптимізації маємо результати, наведені у табл. 2.5.

Таблиця 2.5. Оптимальні розв'язки системи (2.10) за $s = 4$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$	Коефіцієнт ущільнення $U_{70}^{(4)}(P_A)$
3	5	10	$5 \to 3$	1,250
2	3	16	$2 \to 3$	1,125
6	7	38	$7 \to 6$	1,167

В результаті ми отримали три режими, два з яких повністю задовольняють десятковому та шістнадцятковому, тоді як текстовий режим матиме потужність алфавіту 38 замість очікуваного 37. Такий результат пов'язаний з коефіцієнтами у лінійній функції корисності, за якими більшу вагу мають коефіцієнт ущільнення та кількість символів у початковій підпослідовності. Потужність алфавіту в цьому разі є менш принциповою, оскільки можна легко доповнити алфавіт одним додатковим символом, тоді як за іншими параметрами цей розв'язок дає більшу ефективність.

Таким чином, очікується отримати ущільнення 25% для десяткових послідовностей, майже 13% – для шістнадцяткових послідовностей та майже 17% – для текстових послідовностей. Середній відсоток ущільнення складатиме 18%.

Розглянемо тепер цілочислові розв'язки системи (2.7) за $s = 5$. Повна множина допустимих розв'язків наведена у таблиці Г.2 Додатку Г. Для $s = 5$ матимемо $P_\Omega = 3^s = 243$. Тоді $P_{\Omega_{inf}} = P_\Omega - P_{\Omega_{aux}} = 232$, а отже – ДШК-позначка максимально може складатись з 243 ШК-знаків.

Тоді система (2.7) матиме наступний вигляд:

$$\begin{cases} P_A^n \le 232^m \\ n \log_3 P_A > 5m \end{cases} \qquad (2.15)$$

У табл. Г.5 наведено фрагмент таблиці Г.2, яка містить всі допустимі розв'язки системи (2.15).

На рис. Г.2 наведено залежність коефіцієнту ущільнення від потужності алфавіту.

Проаналізуємо наведені розв'язки. Цього разу проведемо експертну оцінку отриманих розв'язків. Як й для $s = 4$, нас цікавитиме знаходження алфавітів, які б відповідали чотирьом стандартним режимам. Відповідно, на найпильнішу увагу заслуговують потужності алфавітів, які покривають алфавіти десяткових та шістнадцяткових цифр, латинський та/або кириличний алфавіти, а також алфавіт ASCII. В текстові алфавіти, як й в попередньому випадку, доцільно також включити найбільш використовувані спеціальні символи, зокрема знаки пунктуації.

Проаналізуємо наведені розв'язки. Цього разу проведемо експертну оцінку отриманих розв'язків. Як й для $s = 4$, нас цікавитиме знаходження алфавітів, які б відповідали чотирьом стандартним режимам. Відповідно, на найпильнішу увагу заслуговують потужності алфавітів, які покривають алфавіти десяткових та шістнадцяткових цифр, латинський та/або кириличний алфавіти, а також алфавіт ASCII. В текстові алфавіти, як й в попередньому випадку, доцільно також включити найбільш використовувані спеціальні символи, зокрема знаки пунктуації.

Проаналізуємо наведені розв'язки. Цього разу проведемо експертну оцінку отриманих розв'язків. Як й для $s = 4$, нас цікавитиме знаходження алфавітів, які б відповідали чотирьом стандартним режимам. Відповідно, на найпильнішу увагу заслуговують потужності алфавітів, які покривають алфавіти десяткових та шістнадцяткових цифр, латинський та/або кириличний алфавіти, а також алфавіт ASCII. В текстові алфавіти, як й в попередньому випадку, доцільно також включити найбільш використовувані спеціальні символи, зокрема знаки пунктуації.

Для $s = 5$ зазначеним вище умовам найкраще задовольняють наступні алфавіти: A_{10} потужністю 10, який повністю покриває десяткові цифри; A_{28} потужністю 28, що покриває шістнадцяткові цифри; A_{82} потужністю 82, який покриває український алфавіт (великі та малі літери), включаючи найуживаніші знаки пунктуації, та A_{128} потужністю 128, він покриває базовий алфавіт ASCII.

Варто зазначити, що хоча у множині розв'язків існують алфавіти потужністю 16, вони нам пасують через те, що мають занизький коефіцієнт ущільнення, а також завеликі значення параметру n – кодувати підпослідовності довжиною у 11-19 символів в загальному сенсі є малоефективним з практичної точки зору, адже довгі шістнадцяткові послідовності можуть бути властиві лише дуже специфічним галузям. Найближчий алфавіт, який дозволяє перетворювати послідовності довжиною 3, 7 та 8, – це обраний нами алфавіт потужності 28.

У табл. Г.6 наведено обрані розв'язки системи (2.7), які будуть розглядатись в подальшому для $s = 5$.

У таблиці подані конкретні типи перетворення для кожного з алфавітів, які видаються найрелевантнішими, однак кожен з них може перетворювати більшу кількість підпослідовностей. Наприклад, у алфавіті A_{82} можна перетворювати послідовності з 7-11 символів, окрім наведених у таблиці 6 символів. Таким чином, у разі, якщо послідовність для перетворення виявиться довшою ніж очікувано (наприклад, остання підпослідовність тексту може складатись не з 6, а з 10 символів), вона буде перетворена з ущільненням, хоча й з меншим, аніж у разі кодування послідовності з 6 символів. Те саме стосується й інших алфавітів, що робить процедуру ущільнення вхідних послідовностей гнучкішою.

Якщо порівняти ступінь ущільнення, отриманого в результаті використання структурного методу, запропонованого у цій дисертаційній

роботі, зі статистичними методами на основі кодів Хаффмана або кодів Шеннона-Фано, то можна побачити, що структурний метод дозволяє досягати підвищення інформаційної щільності подання даних на носії у середньому в 1,6 разів, що переважає аналогічний показник у разі застосування статистичних методів. Результати порівняння ступеня ущільнення для запропонованого структурного методу та для ущільнення із використанням архіватора WinRAR наведені у табл. Г.12 Додатку Г.

З порівняння видно, що структурний метод дає набагато краще ущільнення, коли застосовується до невеликих алфавітно-цифрових послідовностей розміром до 1400 байтів, тоді як при використанні WinRAR ущільнення проявляється для даних розміром від 300 байтів та має нижчі показники, ніж структурний метод.

2.4. Побудова штрихкодових позначок

Розглянемо принцип побудови двовимірної штрихкодової позначки, описаної у цьому розділі.

ДШК-позначка формується з ШК-знаків у вигляді трійкової матриці, на основі якої зрештою відбудеться графічне відображення штрихового коду. Кожна комірка матриці відповідає одному біту ШК-знаку.

Розглянемо принцип побудови двовимірної штрихкодової позначки, описаної у цьому розділі.

Нижче мовою C# наведені фрагменти програмної реалізації бібліотеки `3GCCodeLibrary` (3GC – Three Grades of Color, тобто код з трьома градаціями кольору), яка містить методи перетворення вхідних даних на трійкові послідовності та методи візуалізації результуючої ДШК-позначки. Повний програмний код подано у Додатку Д.

У лістингу 2.2 наведено фрагмент класу `Matrix`, який подає трійкове представлення ДШК-позначки. Це базовий клас, який формує трійкову матрицю ДШК, якій відповідатиме графічне зображення ДШК-позначки.

Лістинг 2.2. Фрагмент опису класу `Matrix`, який подає трійкове представлення ДШК-позначки

```
public class Matrix
{
    public int[,] BarcodeMatrix { get; private set; }
    public int CountOfPatterns { get; private set; }
    public int Height { get; private set; }
    public int Width { get; private set; }
    public int PatternHeight { get; private set; }
    public int PatternWidth { get; private set; }
    public int DigitCapacity { get; private set; }
    public PatternFormat PatternFormat { get; private set; }

    public Matrix(int digitCapacity, int countOfPatterns, PatternFormat patternFormat)
    {
        DigitCapacity = digitCapacity;
        CountOfPatterns = countOfPatterns;
        PatternFormat = patternFormat;

        CalculateMatrixSize(numberOfPatterns);
        BarcodeMatrix = new int[Height, Width];
    }


    public void CalculateMatrixSize(int countOfPatterns)
    {
        // Програмний метод, що обчислює розмір матриці (у комірках)
    }


    public void GetSizeOfPattern()
    {
        // Програмний метод, що обчислює розмір ШК-знаку (у комірках)
    }
```

У лістингу 2.3 наведено фрагмент класу `Symbol`, який забезпечує графічне представлення ДШК-позначки. Як одне з полів, він містить матрицю типу `Matrix` для отримання інформації про трійкове представлення ДШК.

Лістинг 2.3. Фрагмент опису класу `Symbol`, що забезпечує графічне представлення ДШК

```
public class Symbol
{
    private int x0, y0;
    private readonly int cellLength;
    private readonly Matrix matrix;
    public LinkedList<Cell> BarcodeCellCoordinates { get; private set; }
        = new LinkedList<Cell>();
    public Symbol(Matrix matrix, int x0, int y0, int cellLength)
    {
        this.matrix = matrix;
        this.x0 = x0; this.y0 = y0;
        this.cellLength = cellLength;

        BuildBarcodeSymbol();
    }
    private void BuildBarcodeSymbol()
    {
        // Метод, що будує графічне представлення ДШК-позначки
    }
    public void Draw(Graphics drawingArea)
    {
        // Метод, що «відмальовує» відповідні комірки ДШК-позначки
    }
}
```

Спосіб, у який зістиковуються ШК-знаки, залежить від розрядності s знаку, відповідно до якої він може приймати різні форми. На рис Г.3 наведено можливі конфігурації ШК-знаку та, відповідно, способи стикування двох ШК-знаків розрядності $s = 6$. Для кожної конфігурації розроблюється окремий алгоритм заповнення трійкової матриці ДШК-позначки. Для експериментальних досліджень, які проводились в рамках цієї роботи, було розроблено та реалізовано алгоритми для конфігурації (а). Відповідний програмний код наведено у Додатку Д.

Варто також зауважити, що залежно від різновиду ДШК: стандартний код чи дворівневий код – один біт ШК-знаку може бути використаний як контрольний біт для роботи з дворівневими ДШК-позначками. Тоді фактична розрядність ШК-знаку дорівнюватиме $s-1$, що впливатиме на обчислення під час процедури ущільнення даних, але не позначиться на візуальному відображенні позначки.

Структура загальної ДШК-позначки, зображена на рис. Г.4, складається з трьох компонентів: трійкова матриця ШК-знаків, маркер та мірні лінійки.

Маркер та мірні лінійки, які являють собою верхній та лівий бережки графічної позначки, використовуються для позиціонування пристрою зчитування та однозначного визначення орієнтації ДШК-позначки відносно сканера. Маркер розміщується у правому нижньому куті та, залежно від конфігурації ШК-знаку, може являти собою один або два ШК-знаки. Мірні лінійки також задають розміри комірок і розмір (у комірках) ДШК-позначки за горизонталлю та вертикаллю. Сторони комірки завжди мають однаковий розмір, тоді як довжина та висота усієї позначки в загальному випадку є різною. Розмір ДШК-позначки залежить від кількості ШК-знаків, їхньої розмірності й форми та визначається за алгоритмом, реалізація якого наведена у лістингу 2.4.

Кожний ШК-знак як елемент трійкової матриці містить трійкове значення, яке на етапі графічного відображення матриці стає коміркою однієї з трьох градацій кольору. У випадку, який розглядається у цій роботі, це чорний, сірий та білий. На рис. Г.5 показано порядок бітів у ШК-знаку та його вигляд як елемента трійкової матриці та як складової графічної позначки.

Лістинг 2.4. Програмний метод визначення розміру ДШК-позначки (у комірках)

```
public void CalculateMatrixSize(int numberOfPatterns)
{
    double idealNumber = numberOfPatterns / 2 + 1;
    int firstSide = (int)Math.Ceiling(Math.Sqrt(idealNumber));
    int secondSide = (int)Math.Ceiling(idealNumber / firstSide);

    Height = firstSide > secondSide ? secondSide : firstSide;
    Width = firstSide > secondSide ? firstSide : secondSide;

    GetSizeOfPattern();

    Height++;
```

```
    Width++;
}
```

Варто зауважити, що у разі, якщо у ДШК-позначці існують «порожні» ШК-знаки, застосовується службовий ШК-знак – заповнювач PAD. Його трійковий код залежить від розрядності *s* та потужності робочих алфавітів.

2.5. Метод дворівневого штрихового кодування

Дворівневою двовимірною штрихкодовою позначкою (ДДШК-позначкою) називатимемо таку сукупність штрихкодових знаків, яка містить два незалежних одне від одного набори даних.

У загальному випадку, йдеться про будь-які два набори текстових даних, які можуть бути представлені символами ASCII. Кожний з цих двох наборів даних розміщується на окремому рівні ДДШК-позначки. Для того, щоб дати визначення терміну «рівень», введемо декілька понять.

Відповідно до термінології, введеної у п. 2.2, під коміркою розумітимемо одну фізичну комірку (одиничний елемент) матриці. Штрихкодовий знак (ШК-знак) являє собою сукупність суміжних комірок. Тоді *нижнім рівнем* ДДШК-позначки називатимемо множину ШК-знаків, які містять перший набір даних. У цьому розділі як перший набір даних розглядатимемо текстову інформацію, подану латинським алфавітом з використанням деяких спеціальних символів.

На верхньому рівні один ШК-знак розглядається як структурний елемент більшого розміру – квадрант, який фактично є одиничним елементом верхнього рівня. Квадранти об’єднуються у сукупності, які відіграють роль штрихкодових знаків верхнього рівня. Назвемо їх ВШК-знаками (верхніми штрихкодовими знаками). Тоді *верхнім рівнем* ДДШК-позначки називатимемо множину ВШК-знаків, яка містить другий набір

даних. У цьому розділі під другим набором даних розумітимемо цифрову інформацію, подану алфавітом десяткових цифр.

На рис. Г.6 наведено схематичний приклад структури ДДШК-позначки для випадку, коли кількість розрядів квадранту складає $s_q = 9$.

Зауважимо, що коли йдеться про ДДШК-позначку, потрібно розрізняти розрядність s ШК-знаку нижнього рівня, розрядність s_q квадранту верхнього рівня та розрядність s_{up} ВШК-знаку верхнього рівня.

Розрядність s є фактичною розрядністю при кодуванні першого набору даних ДДШК-позначки відповідно до методу ущільненого подання даних, описаного у п. 2.2., тоді як розрядність s_q має на один розряд більше й може бути представлена як $s_q = s + 1$. Цей додатковий біт ігнорується або є відсутнім (залежно від особливостей програмної реалізації), коли здійснюються обчислення для перетворення текстових даних на ШК-знаки. Відповідно, якщо $s_q = 9$, то фактична розрядність одного ШК-знаку, що містить текстову інформацію, складає $s = 8$ й всі дії здійснюються для цієї розрядності. Додатковий біт з'являється при формуванні квадранту на верхньому рівні, коли другий набір даних закодовується за підходом, описаним нижче.

Розрядність s_{up} ВШК-знаку верхнього рівня не залежить напряму від s і s_q, але обирається з урахуванням s_q та кількості ШК-знаків.

В рамках цього дослідження було розроблено програмний засіб для створення ДДШК з трьома градаціями кольору. Нижче подаватимуться фрагменти цього програмного забезпечення, наведені мовою C#.

У лістингу 2.5 наведено фрагмент, в якому подані дві структури (класи) для подання ДДШК-позначки: `UpperLayerPattern` – клас, що описує структуру ВШК-знаку та `TwoLayerBarcode` – клас, що описує загальну структуру ДДШК.

Лістинг 2.5. Опис класів `UpperLayerPattern` та `TwoLayerBarcode`

```
public class UpperLayerPattern
{
    public int x0, y0;

    public LinkedList<Pattern> lowerLayerPatterns;
    public int ControlBit { get; private set; }
}

public class TwoLayerBarcode
{
    private int x0, y0;
    private readonly int cellLength;

    private Matrix matrix;
    private LinkedList<UpperLayerPattern> lowerLayerPatterns;
    public LinkedList<Cell> BarcodeCellCoordinates { get; private set; }
        = new LinkedList<Cell>();
}
```

2.5.1. Кодування даних верхнього рівня

Розглянемо процедуру кодування другого набору даних на верхньому рівні ДДШК-позначки. Перший набір вже закодований, відповідно, нижній рівень позначки, утворений з ШК-знаків, вже сформовано.

На першому етапі визначається кількість квадрантів, яка необхідна для того, щоб закодувати другий набір даних. Для цього повинна бути визначена розрядність s_{up} ВШК-знаку. Оскільки призначення другого набору даних – зберігання коротких алфавітно-цифрових послідовностей, наприклад, посилання на запис у віддаленому сховищі даних, доцільно обрати мінімально можливу розрядність, що для цього дослідження становить $s_{up} = 4$. Відповідно, для подання у ДДШК-позначці одного ВШК-знаку необхідно чотири суміжних квадранти.

Для визначеної розрядності експертом предметної галузі має бути обраний алфавіт, який використовуватиметься для подання другого

набору даних, та тип перетворення. Тип перетворення обирається відповідно до формату другого набору даних.

Формат другого набору даних визначається на початку проєктування програмної системи, яка використовуватиме ДДШК. Оскільки другий набір даних являє собою короткі алфавітно-цифрові послідовності, здебільшого можна точно визначити довжину вхідної послідовності, яка буде сталою для всіх вхідних даних другого набору. Виходячи з цього формату та алфавіту, яким може бути задана ця вхідна послідовність, експерт може визначити відповідний тип перетворення (2.5): $n_{up}(P_{A_{up}}) \to m_{up}(P_{\Omega_{up}})$, де індекс up означає, що розглядається другий набір даних, який подається на верхньому рівні.

Тоді загальна кількість квадрантів, потрібних для кодування другого набору даних, визначатиметься за формулою:

$$N_q = s_{up} \cdot N_{p_{up}}, \tag{2.16}$$

де $N_{p_{up}}$ – кількість ВШК-знаків верхнього рівню, тобто кількість символів вхідного повідомлення в символіці верхнього рівня Ω_{up}.

Відповідно, для того щоб закодувати $N_{p_{up}}$ ВШК-знаків, має виконуватись наступна нерівність:

$$N_p \geq N_q, \tag{2.17}$$

де N_p – кількість ШК-знаків нижнього рівня, які формують ДШК-позначку.

Якщо нерівність (2.17) не виконується, другий набір даних не може бути закодований у цій ДШК-позначці.

Якщо (2.17) виконується, але кількість ШК-знаків більша за потрібну кількість квадрантів, зайві ШК-знаки під час роботи з верхнім рівнем ігноруються.

Завершальним кроком першого етапу є перетворення заданої послідовності другого набору даних на трійкову послідовність, кожний розряд якої відповідає одному квадранту ВШК-знаку.

На другому етапі кожному квадранту з N_q квадрантів у відповідність має бути поставлений цифровий еквівалент. Для випадку штрихового коду з трьома градаціями сірого кольору, який розглядається в цьому дисертаційному дослідженні, цифровий еквівалент може приймати одне з трьох трійкових значень: 0 (цифровий еквівалент білого кольору), 1 (сірого кольору) та 2 (чорного кольору).

Цей цифровий еквівалент обчислюється шляхом підрахунку у квадранті кількості комірок білого, сірого та чорного кольорів. При цьому, контрольний біт цього квадранту (старший або молодший розряд) у цих розрахунках участі не бере та за замовчуванням містить значення 0. Тоді цифровим еквівалентом кольору квадранту вважатиметься той, у який забарвлена найбільша кількість комірок, з яких складається квадрант. У табл. 2.6 показані правила підрахунку цифрового еквіваленту квадранту.

Після визначення цифрового еквіваленту кожного квадранту мають бути розглянуті відповідні групи квадрантів, що утворюють ВШК-знак. Для кожної групи виконується перевірка, чи збігаються цифрові еквіваленти квадрантів з трійковою послідовністю відповідного символу з другого набору даних. Послідовність, у якій розглядаються квадранти та ВШК-знаки, визначається на етапі розроблення програмної системи. На рис. Г.7 представлений один з варіантів обходу ДДШК-позначки.

Таблиця 2.6. Правила встановлення цифрового еквіваленту квадранту

Цифровий еквівалент кольору квадранту	Кількість комірок відповідних кольорів у ШК-знаку
0 «білий»	Білих (0) комірок більше 50%
	Білих (0) та сірих (1) комірок однакова кількість, чорних (2) комірок менше 50%
1 «сірий»	Сірих (1) комірок більше 50%
	Білих (0) та чорних (2) комірок однакова кількість
2 «чорний»	Чорних (2) комірок більше 50%
	Чорних (2) та сірих (1) комірок однакова кількість, білих (0) комірок менше 50%

В ідеальному випадку трійкова послідовність, якій відповідає закодований символ, мала б збігатись з цифровими еквівалентами відповідної послідовності квадрантів. Однак на практиці ці значення переважно розбігатимуться, оскільки цифровий еквівалент квадранту залежить від ШК-знаку нижньої послідовності.

Тому з метою встановлення відповідностей між трійковою послідовністю ВШК-знаку та цифровими еквівалентами квадрантів, цифрові еквіваленти повинні встановлюватись відповідно до того, якими вони мали б бути згідно трійкової послідовності. Але оскільки це значення може розбігатись з дійсним цифровим еквівалентом квадранту, спочатку необхідно перевірити, чи збігаються ці два значення.

Розглянемо це на прикладі. Нехай необхідно закодувати на верхньому рівні цифрову послідовність 0101. Ці чотири біти можуть бути представлені у ДДШК-позначці чотирма суміжними квадрантами.

Кожний квадрант вже має цифровий еквівалент відповідно до процедури визначення, наведеної у табл. 2.11. Якщо цей цифровий еквівалент не збігається з відповідним значенням з трійкової послідовності, яку потрібно закодувати, то контрольному біту присвоюється значення 1 або 2, а комірки, з яких складається квадрант,

інвертуються шляхом обчислення суми дійсного трійкового значення комірки зі значенням контрольного біту за модулем три. Правила інвертування наведені у табл. 2.7.

Варто зауважити, що для зменшення кількості операцій доцільно здійснювати обчислення фактично цифрового еквіваленту квадранту та перевірку на його відповідність очікуваному цифровому еквіваленту із подальшою корекцією контрольного біту одночасно, не розбиваючи ці дві процедури на два окремих послідовних етапи.

Таблиця 2.7. Правила визначення контрольного біту

Фактичний цифровий еквівалент квадранту	Очікуваний цифровий еквівалент квадранту	Значення контрольного біту
0 (більше 50%)	1	1
	2	2
0 (однакова кількість 0 та 1)	1	2
	2	1
0	0	0
1 (більше 50%)	0	2
	2	1
1 (однакова кількість 0 та 2)	0	1
	2	2
1	1	0
2 (більше 50%)	0	1
	1	2
2 (однакова кількість 1 та 2)	0	2
	1	1
2	2	0

На рис. Г.8 наведено приклад кодування цифрової послідовності 0101 на верхньому рівні ДШК-позначки. Чотири квадранти, які передбачається використати для кодування цієї послідовності, мають цифрові еквіваленти 2011. Відповідно, у цих двох трійкових

послідовностях збігається лише молодший біт, а отже, три старших біти мають бути інвертовані.

В результаті дослідження всіх груп квадрантів на їх відповідність трійковим послідовностям ВШК-знаків верхній рівень ДДШК-позначки набуває кінцевого вигляду. Тоді під час декодування другого набору даних будуть отримані трійкові значення, які можна розкодувати у початкову алфавітно-цифрову послідовність.

На етапі ж декодування даних нижнього рівня ДДШК-позначки необхідно знову інвертувати кольори комірок цього квадранту шляхом віднімання від значень бітів комірок значення контрольного біту за модулем 3.

Таким чином, розглядаючи ДДШК-позначку як таку, що містить комірки на нижньому рівні та квадранти на верхньому рівні, отримуємо можливість кодувати дві незалежні одна від одної послідовності символів, зберігаючи їх в одній й тій самій графічній ДШК-позначці.

2.5.2. Етапи методу дворівневого штрихового кодування

Сформулюємо тепер етапи, з яких складається метод дворівневого штрихового кодування.

Вхідними даними є дві алфавітно-цифрові послідовності: текстова послідовність для кодування на нижньому рівні та коротка послідовність для кодування на верхньому рівні. Також визначається розрядність ШК-знаків нижнього та верхнього рівнів.

Етап 1. Перетворення текстової алфавітно-цифрової послідовності для подання на нижньому рівні ДДШК-позначки на трійкові кодовектори розрядності s. Спосіб, у який здійснюється перетворення, визначається вимогами конкретної галузі застосування. Результатом є послідовність трійкових кодовекторів.

Етап 2. Перетворення короткої алфавітно-цифрової послідовності для кодування на верхньому рівні на трійкові кодовектори розрядності s_{up}. Спосіб кодування даних може збігатись із обраним для Етапу 1, а може бути обраним інший метод, залежно від вимог галузі застосування. Результатом є послідовність трійкових кодовекторів.

Етап 3. Визначення трійкових еквівалентів квадрантів, які використовуватимуться для кодування даних для подання на верхньому рівні.

Етап 4. Визначення контрольного біту для кожного трійкового кодовектора нижнього рівня за табличним методом відповідно до фактичного та очікуваного значення квадранту.

Етап 5. Кодування даних нижнього рівня згідно правил конвертації цифрових еквівалентів на основі контрольних бітів.

Етап 6. Формування результуючої ДДШК-позначки.

Такий підхід дозволяє забезпечити завадостійкість верхнього рівня ДДШК-позначки – навіть у разі пошкодження контрольного біту дані верхнього рівня можуть бути успішно відновлені. Якщо на верхньому рівні закодований, наприклад, ідентифікатор певного об’єкту обліку для доступу до віддаленої бази даних, можна отримати доступ до повної інформації про цей об’єкт без декодування обмеженої автономної інформації, які зберігалась на нижньому рівні позначки.

Друга перевага цього підходу полягає у тому, що застосування дворівневого кодування інформації дозволяє розмежувати права доступу до даних у системі з різними ролями, що дає можливість захищено зберігати повну інформацію стосовно об’єкту обліку, включаючи, наприклад, фінансову документацію, особисту інформацію про власника об’єкта тощо.

Крім того, дворівневий підхід дозволяє певною мірою підвищити продуктивність системи, дозволяючи швидко отримати доступ до

посилання на віддалену базу даних, не розшифровуючи текст, що міститься у нижньому рівні ДДШК-позначки.

2.6. Дослідження процедури формування дворівневих штрихкодових позначок

Розглянемо приклад формування дворівневої двовимірної штрихкодової позначки розрядністю $s = 8$.

Нехай потрібно закодувати дві послідовності даних: текстову та цифрову. Текстова інформація задана послідовністю завдовжки 24 символи, поданих текстовим алфавітом потужності 82, що містить латинські великі та малі літери, цифри, спеціальні символи та службові символи, як-от кінець рядка та заповнювач:

$$\begin{matrix} UA157\,/\,4 \\ Bratislava - Kyiv \end{matrix} \qquad (2.18)$$

Для розрядності $s = 8$ такий алфавіт може перетворювати 9 алфавітних символів у 5 ШК-знаків з ущільненням 12%.

Цифрова інформація, яка у досліджуваному прикладі є кодом ідентифікації певного об’єкту логістики для пошуку у віддаленій базі даних, задана послідовністю завдовжки 5 символів, поданих десятковим алфавітом потужності 11, який окрім цифр містить символ скісної риски: *157/4*. У цьому алфавіті 5 алфавітних символів перетворюються на 3 ВШК-знаки розрядності $s_{up} = 4$.

На першому етапі потрібно закодувати на нижньому рівні текстову інформацію. Скористаємось для цього методом ущільнення, описаним у п. 2.5.1. В результаті алфавітно-цифрова послідовність (2.18) завдовжки 24

символи перетворюється на послідовність ШК-знаків у системі числення $P_{\Omega_{inf}} = 6550$ завдовжки 13 символів наступного вигляді:

$$22\ 1465\ 3704\ 5075\ 457\ 48\ 747\ 2697\ 327\ 6418\ 60\ 1343\ 97. \quad (2.19)$$

Кожний ШК-знак у системі числення 6550 має тепер бути перетворений на трійкову послідовність, яка формуватиме нижній рівень ДДШК-позначки. Результатом перетворення стає наступна послідовність трійкових ШК-знаків:

$$\begin{array}{l} 00000211\ \ 02000020\ \ 12002012\ \ 20221222\ \ 00121221 \\ 00001210\ \ 01000200\ \ 10200220\ \ 00110010\ \ 22210201 \\ \qquad\qquad 00002020\ \ 01211202\ \ 00010121 \end{array} \quad (2.20)$$

Згідно з процедурою визначення розміру ДШК-позначки, поданої у п. 2.4, визначаємо, що кількість ШК-знаків, з яких можна сформувати прямокутну позначку, складає 15 ШК-знаків. Невикористані два ШК-знаки заповнюються знаками-заповнювачами 00002022.

На рис. Г.9 показано графічний вигляд закодованої текстової інформації (2.18). Наразі контрольний біт кожного ШК-знаку містить значення за замовчуванням – 0.

Наступним етапом маємо закодувати цифрову послідовність *157/4*. Також скористаємось методом ущільнення даних, але цього разу розрядність ВШК-знаків складає $s_{up} = 4$. Відповідно до цього 5 символів цифрової послідовності у системі числення 11 переходять у три ВШК-знаки верхнього рівня у системи числення $P_{\Omega_{up}} = 81$:

$$3\ 31\ 63. \quad (2.21)$$

Маючи тепер закодовану цифрову послідовність, перетворюємо її на трійкові послідовності розрядності 4, які мають наступний вигляд:

$$0010\ 1011\ 2100. \qquad (2.22)$$

Згідно формулам (2.16) та (2.17) перевіряємо, чи може така кількість ВШК-знаків бути закодована у ДШК-позначці, наведеній на рис. 2.11:

$$N_q = s_{up} \cdot N_{p_{up}} = 4 \cdot 3 = 12,$$
$$N_p \geq N_q, \text{ тобто } 15 \geq 12.$$

Отже, закодована цифрова послідовність (2.22) може бути розміщена на верхньому рівні ДДШК-позначки. На рис. Г.10 показано, як, згідно з формою позначки, буде обрано квадранти для кодування цифрової послідовності.

Відповідно, маємо три групи по 4 квадранти. Порядок розміщення цифрових даних (2.22) також наведено на рис. 2.10.

Для подання цифрових даних (2.22) на верхньому рівній цієї ДДШК-позначки визначимо поточні цифрові еквіваленти квадрантів у кожній з трьох груп за табл. Г.5. В результаті маємо таку послідовність трійкових значень:

$$0010\ 2000\ 0022.$$

Порівняємо кожне з отриманих трійкових значень із відповідними ВШК-знаками (2.20). Легко побачити, що перший ВШК-знак збігається з фактичним вмістом відповідних квадрантів ДДШК-позначки. Відповідно,

значення контрольних бітів кожного з цих квадрантів залишаються за замовчуванням дорівнювати 0.

Проте трійкові значення другого та третього ВШК-знаків з фактичними значеннями квадрантів розбігаються: 2000 замість 1011 (збігається лише другий розряд) та 0022 замість 2100 (всі розряди розбігаються). Отже, маємо скорегувати контрольним бітом сім з восьми квадрантів.

У табл. 2.8 наведено обрання контрольного біту залежно від фактичного та очікуваного значення розряду.

Після зміни контрольних бітів здійснюємо операцію додавання контрольного біту до кожного розряду відповідного квадранту за модулем 3.

Таблиця 2.8. Обрання контрольних бітів

Фактичний цифровий еквівалент	Очікуваний цифровий еквівалент	Контрольний біт
2 0 0 0 → 1 0 1 1		
2 (однакова кількість 2 та 1)	1	1
0 (більше 50%)	0	0
0 (більше 50%)	1	1
0 (більше 50%)	1	1
0 0 2 2 → 2 1 0 0		
0 (більше 50%)	2	2
0 (більше 50%)	1	1
2 (більше 50%)	0	1
2 (однакова кількість 2 та 1)	0	2

Таким чином, всі квадранти приходять у відповідність з трійковими послідовностями (2.20), що дозволяє стверджувати про успішне кодування цифрової послідовності *157/4* на верхньому рівні ДДШК-позначки, фінальний вигляд якої наведено на рис. Г.11.

Для декодування верхнього рівню та отримання вихідної символьно-цифрової послідовності обчислюється цифровий еквівалент кожного квадранту відповідного ВШК-знаку. Використовуючи декодовані дані, можливий доступ до відповідного запису у віддаленому сховищі даних для отримання повної інформації про об’єкт логістики.

Щоб декодувати нижній рівень автономних даних, потрібно здійснити обернену операцію для інвертування за модулем 3 значень усіх комірок позначки, після чого можна отримати початкову текстову інформацію (2.18), яка міститься на нижньому рівні ДДШК-позначки.

2.7. Висновки до другого розділу

1. Розроблено процедуру ущільнення вхідної послідовності для подання її у вигляді штрихового коду з трьома градаціями кольору, що дозволяє визначити оптимальні параметри для отримання максимального коефіцієнта ущільнення за заданої потужності алфавіту.
2. Розроблено процедуру багатокритеріальної оптимізації для визначення потужності алфавіту, що дозволяє формувати набори символів, що відповідають вимогам користувачів програмної системи автоматичної ідентифікації об’єктів логістики.
3. Удосконалено алгоритмічне забезпечення та розроблено і досліджено програмне забезпечення процесів формування даних у вигляді двовимірних штрихових кодів, яке, на відміну від відомих, ґрунтується на використанні процедури виконання операцій над даними у трійковій системі числення для формування штрихкодової позначки у трьох градаціях кольору та процедури багатокритеріальної оптимізації для визначення потужності алфавіту, що уможливлює формування символіки штрихового коду відповідно до вимог галузі застосування та підвищення щільності подання даних у 1.4–1.7 разів.

4. Розроблено процедуру формування дворівневої штрихкодової позначки, до дозволяє подавати в одному штрихкодовому зображенні два набори даних різних типів та довжини, що дозволяє забезпечити компактне подання даних та розмежування доступу до інформації.
5. Розроблено та досліджено алгоритмічне та програмне забезпечення процесів формування дворівневих штрихкодових позначок з трьома градаціями кольору, яке, на відміну від відомих, ґрунтується на використанні процедури визначення контрольного біту та процедури формування дворівневої штрихкодової позначки у трьох градаціях кольору, що уможливлює два рівні доступу до інформації про об’єкт логістики.

РОЗДІЛ 3. АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЗАВАДОСТІЙКОСТІ ШТРИХОВИХ КОДІВ З ТРЬОМА ГРАДАЦІЯМИ КОЛЬОРУ

3.1. Задача забезпечення завадостійкості штрихкодових зображень

Вхідна алфавітно-цифрова послідовність після ущільнення перетворюється на послідовність інформаційних ШК-знаків. Ця послідовність може бути візуалізована як ДШК-позначка з трьома градаціями кольору та нанесена на носій у вигляді графічного зображення. Відповідно, у загальному випадку кодування ущільненої послідовності може відбуватись без використання завадостійкого коректувального коду. Втім, така ДШК-позначка не буде завадостійкою, а отже, будь-яке спотворення цього зображення, навіть мінімальне, призводитиме до виникнення помилок під час зчитування та розпізнавання ШК-знаків позначки.

Практика використання штрихових кодів свідчить про те, що штрихкодова інформація має бути завадозахищеною для уникнення помилок під час процедури декодування штрихкодових даних. Тому додаткове кодування послідовності ШК-знаків завадостійким коректувальним кодом повинно передувати виготовленню графічного зображення ДШК-позначки та його нанесення на носій.

Таким чином, наступним етапом створення штрихового коду з трьома градаціями кольору для подання алфавітно-цифрових даних є завадостійке кодування, яке ґрунтується на використанні коректувального коду, здатного виправляти багатократні спотворення. Цей процес передбачає обчислення за певними правилами контрольних кодовекторів, які додаються до інформаційних, в результаті чого утворюється кодове слово коректувального коду. Кодовому слову відповідає завадостійка

штрихкодова позначка, яка, незважаючи на можливі ушкодження поверхні носія, може бути коректно зчитана та оброблена.

Завадостійкість є важливою властивістю штрихових кодів. Завадостійким штриховим кодом називатимемо такий, що дозволяє виявляти та виправляти помилки зчитування, які можуть виникати внаслідок ушкодження окремих ШК-елементів ШК-позначки. У п. 1.3.2 здійснено аналіз можливих ушкоджень елементів у штрихових кодах, який дозволяє зробити висновок про те, що для забезпечення надійного відтворення даних з штрихкодової позначки необхідно перетворену алфавітно-цифрову послідовність кодувати таким завадостійким кодом, який дозволяє виправлення ушкоджень двох видів – помилок та стирань.

В цьому розділі розглянемо алгоритмічні підходи до забезпечення завадостійкості штрихових кодів з трьома градаціями кольору.

3.2. Алгоритмічне та програмне забезпечення процесу завадозахищеного штрихового кодування з трьома градаціями кольору

Для забезпечення завадостійкості даних, поданих у вигляді штрихових кодів, розглядатимемо ШК-знаки, на які перетворюються вхідні алфавітно-цифрові послідовності, як розряди послідовності всіх ШК-знаків, а послідовність ШК-знаків, з якої й формується ДШК-позначка, кодуватимемо за допомогою певного коректувального коду для виправлення ушкоджень двох типів – помилок та стирань, які було виявлено в результаті аналізу можливих ушкоджень у п. 1.3.2.

Під помилкою в цьому контексті розумітимемо такий тип спотворення, коли один ШК-знак перетворюється на інший ШК-знак внаслідок певного ушкодження.

Стиранням називатимемо спотворення таке, що декодувальне програмне забезпечення не може зіставити зчитаний з носія ШК-знак зі жодним з ШК-знаків символіки, тобто було зчитано такий знак, якого немає в алфавіті символіки, яка використовується у цьому штриховому коді.

При проєктуванні системи контролю спотворень важливо забезпечити наявність декількох рівнів корекції, оскільки залежно від умов зберігання об’єктів, промаркованих ДШК-позначками, якості носія штрихкодової інформації тощо кількість можливих спотворень може варіюватись.

Таким чином, метою цього розділу є проєктування таких засобів забезпечення завадостійкості штрихкодових позначок двовимірних штрихових кодів з трьома градаціями кольору, які забезпечать виправлення потенційних багатократних спотворень, а саме – помилок та стирань, та уможливлять гнучке налаштування рівнів захисту перед безпосереднім формування ДШК-позначки.

3.2.1. Обрання коректувального коду та алгоритмічних засобів забезпечення завадостійкості

Як було визначено вище, для забезпечення надійності зберігання інформації у вигляді штрихкодової позначки послідовність ШК-знаків, отриману в результаті кодування вихідних даних з ущільненням, алгоритмічне та програмне забезпечення чого описано у розділі 2, необхідно закодувати, використовуючи певний коректувальний код, за допомогою якого можна виправляти багатократні спотворення.

На сьогодні розроблено низку коректувальних кодів, які можуть виправляти багатократні помилки [117, 118, 121, 128, 142, 144–146, 148], зокрема, це код Боуза-Чоудхурі-Хоквінгема, код Ріда-Соломона, код

Юстесена, код Гоппи, циклічні коди, згорткові коди тощо. Втім, для забезпечення завадостійкості ДШК-позначок в цьому дослідженні був обраний коректувальний коду Ріда-Соломона. Проаналізуємо особливості цього коду, які привели до обрання саме цього коректувального коду.

В першу чергу варто зауважити, що код Ріда-Соломона є многозначним кодом, тоді як здебільшого коректувальні коди є двійковими. Відповідно, завдяки цій властивості, код Ріда-Соломона послуговується не бітами, а символами, які належать до певного многозначного алфавіту.

Для двовимірних штрихових кодів з трьома градаціями кольору роль таких многозначних символів виконують ШК-знакі, що відповідають порядковим номерам $0, 1, \ldots, P_\Omega - 1$ алфавіту Ω. Це уможливлює виправлення ШК-знаку цілком, замість корегування його окремих частин – трійкових елементів.

Другою особливістю на користь коду Ріда-Соломона є його приналежність до категорії так званих досконалих кодів. Досконалі коди мають мінімально можливу надлишковість для заданої коректувальної здатності [117, 118, 121, 128, 142, 144–146, 148]. Серед коректувальних кодів лише чотири з них є досконалими, а саме – код з контролем на парність-непарність, код Хемінга, (23,12)-код Голея та код Ріда-Соломона, й серед них лише код Ріда-Соломона є многозначним кодом, що є принциповим для цього дослідження.

Ще однією перевагою кодів Ріда-Соломона є низька обчислювальна складність процедури кодування-декодування, особливо порівняно з іншими коректувальними кодами.

Таким чином, застосування коректувального коду Ріда-Соломона зумовлюється сформульованими вище особливостями, які забезпечують виправлення спотворень двох типів – помилок та стирань.

3.2.2. Основні поняття коректувальних кодів Ріда-Соломона

Сформулюємо основні засади завадостійкого кодування із використанням коректувального коду. Для цього розглянемо деякі базові поняття [117, 118, 121, 128, 142, 144–146, 148].

Мінімальною відстанню коректувального коду називають мінімальну кількість розрядів, при спотворені яких один дозволений кодовий вектор переходить в інший дозволений кодовий вектор. Якщо ця відстань дорівнює d, то будь-яка конфігурація з δ стирань може бути відновлена тоді, коли $d \geq \delta + 1$. Водночас коректувальний код з мінімальною відстанню d дозволяє виправляти будь-яку конфігурацію з γ помилок та δ стирань за умови, що:

$$d \geq 2\gamma + 1 + \delta. \tag{3.1}$$

Застосуємо коректувальний код Ріда-Соломона з мінімальною відстанню $d = n - k + 1$, де n – загальна довжина послідовності ШК-знаків, а k – кількість інформаційних ШК-знаків у цій послідовності, для забезпечення завадостійкості ДШК-позначок. Тоді кількість контрольних ШК-знаків складатиме $r = n - k$. Підставивши d та r у (3.1), отримаємо нерівність вигляду:

$$r \geq 2\gamma + \delta. \tag{3.2}$$

Відповідно, якщо потрібно виправляти, наприклад, $\gamma = 2$ та $\delta = 3$ стирання, то для корекції спотворень кодова послідовність коду Ріда-

Соломона повинна складатись з щонайменше $r = 2\gamma + \delta = 2 \cdot 2 + 3 = 7$ контрольних ШК-знаків.

Нехай система контроля спотворень має $\Lambda + 1 = \{0, 1, \ldots, \Lambda\}$ рівнів корекції спотворень. Якщо на етапі формування завадостійкої ДШК-позначки був обраний нульовий рівень корекції, це означає, що ця позначка не міститиме контрольних ШК-знаків й, відповідно, складається лише з інформаційних ШК-знаків.

Для того, щоб на рівні Λ забезпечити виправлення будь-якого спотворення, необхідно, щоб виконувалась нерівність:

$$2\gamma + \delta \leq 2^{\Lambda} - 1. \tag{3.3}$$

Можливі й інші видозміни відношення між Λ та величинами γ і δ, наприклад, $2\gamma + \delta \leq 2^{\Lambda+1} - 2$, $2\gamma + \delta \leq 2^{\Lambda+1}$, $\gamma + \delta \leq 2^{\Lambda+1}$ тощо.

Тоді, наприклад, якщо при формуванні певної ДШК-позначки ми оберемо рівень корекції спотворень такий, що дорівнює $\Lambda = 2$, то послідовність ШК-знаків коду Ріда-Соломона міститиме 3 контрольні ШК-знаки, що уможливлять виправлення наступних конфігурацій помилок γ та стирань δ: $\gamma = 1, \delta = 1$ та $\gamma = 0, \delta = 3$. Повний список рівнів корекції спотворень та їхніх характеристик наведено у табл. Є.1.

Обрання того чи іншого рівня корекції зумовлюється декількома факторами. По-перше, це залежить від ємності ДШК-позначки. Що більше ШК-знаків вона може містити, тим вищій рівень корекції варто задати. У табл. Є.2 наведено загальні рекомендації щодо рівнів корекції спотворень залежно від кількості інформаційних ШК-знаків.

По-друге, потрібно брати до уваги середовище, у якому здійснюватиметься використання ДШК-позначки. Залежно від ступеню ризиків виникнення спотворень обирається вищий або нижчий рівень

корекції. Так, наприклад, позначка, що зберігається у вигляді цифрового файлу з меншою ймовірністю буде пошкоджена, порівняно з позначкою на паперовому носії, якою промарковано фізичний об'єкт, що планується транспортувати, адже у середовищі вантажівки цілком можливе істотне ушкодження носія ДШК-позначки.

Максимальній рівень корекції можна визначити з наступного співвідношення [117, 118, 121, 128, 142, 144–146, 148]:

$$2\Lambda - 1 = \frac{2}{3}k, \tag{3.4}$$

де k – кількість інформаційних ШК-знаків.

Втім, за потреби кількість контрольних ШК-знаків може бути збільшено.

3.2.3. Процедура завадостійкого кодування на основі коректувального коду Ріда-Соломона

Розглянемо алгоритмічне та програмне забезпечення процедури завадостійкого кодування на основі коректувального коду Ріда-Соломона.

Потужність символіки будь-якого двовимірного штрихового коду складає P_Ω. У загальному випадку $P_\Omega = p^s$, де p – просте число, однак у подальших обчисленням розглядатимемо $P_\Omega = 3^6$ для ДШК-кодів з трьома градаціями кольору. Тоді інформаційні та контрольні ШК-знаки символіки Ω належатимуть полю Галуа $GF(P_\Omega)$, тобто полю $GF(3^6)$.

Відповідно до процедури кодування даних коректувальним кодом Ріда-Соломона [117, 118, 121, 128, 142, 144–146, 148] потрібно:

1) задати рівень Λ корекції спотворень;

2) визначити кількість r контрольних ШК-знаків за (3.2) та (3.3);

3) отримати кодовий поліном $c(x)$;

4) кожному ШК-знаку послідовності $c_{n-1}\, c_{n-2} \ldots c_1\, c_0$, де c_i – коефіцієнти кодового поліному $c(x)$, поставити у відповідність трійкову послідовність та сформувати ДШК-позначку.

Кодовий поліном коректувального коду Ріда-Соломона має наступний вигляд:

$$c(x) = a(x)g(x) = c_{n-1}x^{n-1} + \ldots + c_1 x + c_0, \tag{3.5}$$

де $a(x)$ – інформаційний поліном,

$g(x)$ – твірний поліном коректувального коду Ріда-Соломона,

n – кількість ШК-знаків у ДШК-позначці, $n = k + r$.

Інформаційний поліном $a(x)$ вигляду:

$$a(x) = \sum_{i=0}^{k-1} a_i x^i \tag{3.6}$$

формується з інформаційних ШК-знаків $a_i \in 0, 1, \ldots, P_{\Omega_{inf}} - 1$, тобто порядкових номерів ШК-знаків в алфавіті Ω_{inf}.

Наприклад, інформаційний поліном послідовності з шести ШК-знаків: 3 674 40 26 586 670, якій відповідає текстовий рядок *Sikorsky*, матиме вигляд:

$$a(x) = 3x^5 + 674x^4 + 40x^3 + 26x^2 + 586x + 670. \tag{3.7}$$

Твірний поліном коректувального коду Ріда-Соломона складається з добутку наступного вигляду:

$$g(x) = (x - \alpha)(x - \alpha^2)\ldots(x - \alpha^r), \tag{3.8}$$

де α – примітивний елемент поля $GF(P_\Omega)$,

r – кількість контрольних ШК-знаків.

Примітивним елементом α називатимемо таке число, степені (від 0 до $n-1$) якого за модулем незвідного полінома формують множину всіх ненульових елементів поля. Наприклад, якщо $P_\Omega = 729$, то $\alpha = 3$.

Оскільки P_Ω є степенем простого числа, то маємо побудувати розширення $GF(3^6)$ над полем Галуа $GF(3)$. Для цього скористаємось незвідним поліномом степеню $s = 6$, коефіцієнти якого є елементами поля $GF(3)$. Існує 60 незвідних поліномів степеню 6 [117, 118, 121, 128, 142, 144–146, 148], й обрати для побудови розширення $GF(3^6)$ можна будь-який з них. В цьому дослідженні візьмемо незвідний поліном $m_6(x) = x^6 + x + 2$.

Елементи розширення $GF(3^6)$ над полем $GF(3)$ можна подати у декількох формах:

– у вигляді невід'ємного та від'ємного степенів α;

– у вигляді полінома від α;

– у трійковому вигляді;

– у десятковому вигляді.

В рамках цього дослідження було розроблено програмний засіб для завадостійкого кодування на основі коректувального коду Ріда-Соломона. Нижче подаватимуться фрагменти цього програмного забезпечення, наведені мовою C#.

Для отримання повного подання елементів поля $GF(p^s)$ кожний елемент представлятимемо як структуру (клас) `GFElement`, яка містить чотири поля (Properties): `AlphaPower` – подання елемента у вигляді невід'ємного степеню α, `NegativeAlphaPower` – подання елемента у вигляді від'ємного степеню α, `TernaryRepresentation` – подання елемента у вигляді трійкового числа, `DecimalRepresentation` – подання елемента у вигляді десяткового числа. Створювати окреме поле для подання елемента у вигляді полінома від α немає сенсу, оскільки він може легко бути отриманий з подання у вигляді трійкового числа: $t_{s-1}x^{s-1} + t_{s-2}x^{s-2} + \ldots + t_1 x + t_0$, де t_i – i-й розряд трійкового s-розрядного числа.

Опис класу `GFElement` представлений у лістингу 3.1.

Лістинг 3.1. Опис класу `GFElement` для подання елемента поля $GF(p^s)$

```
public class GFElement
{
    public int AlphaPower { get; set; }
    public int NegativeAlphaPower { get; set; }
    public string TernaryRepresentation { get; set; }
    public int DecimalRepresentation { get; set; }
}
```

Повне подання елементів поля $GF(p^s)$ отримується шляхом ділення поліному α^i, де $i = \{0, 1, \ldots, p^{s-1}\}$, на обраний незвідний поліном $m(\alpha)$.

Для здійснення математичних операцій над поліномами зараз і в подальшому було розроблено декілька структур (класів), за допомогою яких можна представити будь-який поліном від x або від α. У лістингу 3.2 наведено опис класів, якими ми послуговуватимемось для представлення поліномів у програмному вигляді.

Лістинг 3.2. Опис класів, що формують структуру поліному

```
public class PolynomialTerm
{
    public int Power { get; set; }
    public int Coefficient { get; set; }
}
public class PolynomialMixedTerm
{
    public PolynomialTerm TermCoefficient { get; set; } = new PolynomialTerm();
    public int Power { get; set; }
}
public class Polynomial
{
    public List<PolynomialTerm> xTerms;
    public List<PolynomialTerm> alphaTerms;
    public List<PolynomialMixedTerm> xAlphaTerms;
}
```

Доданки поліному можуть бути двох типів (`PolynomialTerm` та `PolynomialMixedTerm`), залежно від того, містить доданок степінь x чи тільки степінь α. Якщо доданок не містить степеню x, від подається структурою (класом) `PolynomialTerm`, який складається з полів `Power` – степінь α та `Coefficient` – трійковий коефіцієнт при цьому степені.

У разі ж, якщо доданок є «змішаним», тобто містить й α, й x, то він подається структурою `PolynomialMixedTerm`, яка містить поля `TermCoefficient` та `Power` – степінь x. `TermCoefficient` є типу `PolynomialTerm`, оскільки коефіцієнтом при x виступає α у степені.

Тоді структура поліному складається зі списків `xTerms` – зберігає усі доданки x, які не мають коефіцієнта α (здебільшого це один доданок з максимальним степенем x); `alphaTerms` – зберігає всі доданки α, які не залежать від степеню x (здебільшого це один додаток α, степінь x якого дорівнює нулю); та `xAlphaTerms` (доданки вигляду $\alpha^i x^j$).

Всі операції, які виконуються над елементами поля $GF(p^s)$ та над поліномами, здійснюються згідно наступних правил :

1) для здійснення операції додавання елементів поля потрібно послуговуватись поданням цих елементів у вигляді трійкової послідовності із порозрядним додаванням елементів цих послідовностей за

правилами трійкової арифметики (за модулем 3), наприклад, $243+347=\alpha^5+\alpha^{44}=100000+110212=210212=590$;

2) для здійснення операції множення елементів поля потрібно використовувати подання елементів у вигляді невід'ємного степеню примітивного елемента поля, наприклад, $13\cdot 96=\alpha^{12}\cdot\alpha^{22}=\alpha^{34}=409$;

3) для здійснення операцій ділення елементів поля потрібно чисельник і знаменник подати у вигляді невід'ємних степенів, після чого перейти до операції множення та скористатись правилом (2) множення елементів поля, наприклад, $248\div 27=\alpha^{11}\div\alpha^3=\alpha^{11}\cdot\alpha^{-3}=\alpha^8=63$.

Програмне забезпечення операцій над елементами поля, а також операцій множення та ділення поліномів подано мовою C# у Додатку Е.

У табл. Є.3 наведено таблицю подання елементів поля $GF(3^6)$ за модулем незвідного поліному $m_6(x)=x^6+x+2$.

Тоді, маючи таблицю повного подання 729 елементів поля $GF(3^6)$, можемо обчислити твірний поліном (3.8) шляхом множення поліномів $(x-\alpha)(x-\alpha^2)\ldots(x-\alpha^r)$, причому згідно трійкової математики (0 – 1 за модулем 3 дає 2) можемо перейти від операції віднімання до операції додавання, тобто: $g(x)=(x+2\alpha)(x+2\alpha^2)\ldots(x+2\alpha^r)$.

Після розкриття дужок твірний поліном матиме вигляд:

$$g(x)=\sum_{j=0}^{r} g_j x_j, \qquad (3.9)$$

де g_j – елементи поля $GF(p^s)$, причому $g_r=1$.

У полі Галуа $GF(3^6)$ за рівня корекції спотворень $\Lambda = 2$, який в цій роботі використовується у всіх дослідженнях, $r = 3$ згідно з таблицею 3.1. Відповідно, твірний поліном матиме наступний вигляд:

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3) = (x+2\alpha)(x+2\alpha^2)(x+2\alpha^3) =$$
$$= x^3 + (2\alpha^3 + 2\alpha^2 + 2\alpha)x^2 + (\alpha^5 + \alpha^4 + \alpha^3)x + \alpha + 2 =$$
$$= x^3 + \alpha^{377}x^2 + \alpha^{15}x + \alpha^{370} = x^3 + 78x^2 + 351x + 5.$$

Тобто, твірний поліном $g(x)$ у полі $GF(3^6)$ дорівнює:

$$g(x) = x^3 + 78x^2 + 351x + 5. \qquad (3.10)$$

Маючи твірний поліном, можемо тепер обчислити кодовий поліном, коефіцієнти якого й утворять ШК-знаки закодованої вхідної послідовності. Згідно (3.5) для текстової послідовності *Sikorsky*, якому відповідає інформаційний поліном (3.7), отримаємо кодовий поліном вигляду:

$$c(x) = a(x)g(x) = (3x^5 + 674x^4 + 40x^3 + 26x^2 + 586x + 670)\cdot(x^3 + 78x^2 +$$
$$+ 351x + 5) = \alpha^1 x^8 + \alpha^{497}x^7 + \alpha^{148}x^6 + \alpha^{438}x^5 + \alpha^{400}x^4 + \alpha^{95}x^3 +$$
$$+ \alpha^{382}x^2 + \alpha^{533}x + \alpha^{648} = 3x^8 + 638x^7 + 22x^6 + 716x^5 + 58x^4 +$$
$$+ 679x^3 + 29x^2 + 173x + 157.$$

Таким чином, текстовій послідовності *Sikorsky*, якій відповідають шість ШК-символів 3 674 40 26 586 670, поставлено у відповідність кодову послідовність 3 638 22 716 58 679 29 173 157, у якій забезпечується одночасне виправлення однієї помилки та одного стирання або трьох стирань.

3.2.4. Виправлення помилок та стирань при декодуванні штрихкодових даних

В результаті зчитування ДШК-позначки отримується лінійна послідовність ШК-знаків загальною кількістю n, а також r контрольних ШК-знаків, після чого відбувається спроба ототожнення кожний ШК-знаку цієї послідовності зі знаками символіки Ω. Якщо ототожнити не вдалось, то такий ШК-знак вважається стертим. Оскільки порядковий номер цього ШК-знаку у позначці відомий, він фіксується, стаючи так званим локатором спотворення. Значення спотворення при цьому невідоме. На етапі налаштування процедури декодування за Рідом-Соломоном таким стертим ШК-знакам присвоюється певне визначене наперед значення, наприклад, -1, оскільки від'ємне значення ШК-знаку існувати не може.

Відповідно, на етапі декодування зчитаної послідовності мають бути виправлені наявні у ДШК-позначці спотворення двох типів: стирання, для яких відомі локатори і не відомі величини спотворень, та помилки, для яких не відомі ані локатори, ані величини спотворень.

Розглянемо далі теоретичні засади процедури декодування спотворених даних.

Оскільки коректувальний код здатний виправляти спотворення двох типів, то розглядатимемо два поліноми локаторів:

1) поліном локаторів стирань:

$$\lambda(x) = \prod_{l=1}^{\delta} (1 - X_{i_l} x), \tag{3.11}$$

де X_{i_l} – локатори стирань,

δ – кількість стирань;

2) поліном локаторів помилок:

$$\sigma(x) = \prod_{j=1}^{\gamma} (1 - X_{p_j} x), \tag{3.12}$$

де X_{p_j} – локатори помилок,

γ – кількість помилок.

Згідно з [128] декодування послідовності ШК-знаків здійснюється за наступним алгоритмом.

Крок 1. У зчитаній послідовності ШК-знаків необхідно зафіксувати номери $i_1, i_2, \ldots, i_\delta$ розрядів, які містять δ стирань, після чого перевіряється нерівність $\delta \leq r$.

Якщо $\delta > r$, тобто кількість стирань є більшою за коректувальну здатність коду, визначену на етапі налаштувань, то процедура декодування зупиняється, оскільки зчитана послідовність має невиправні спотворення.

Якщо ж нерівність виконується, то необхідно знайти δ локаторів стирань $X_{i_1} = 3^{i_1}$, $X_{i_2} = 3^{i_2}, \ldots, X_{i_\delta} = 3^{i_\delta}$. При цьому, розрядам, в яких були виявлені стирання, мають бути присвоєні значення -1.

Тепер можна знайти поліном локаторів стирань $\lambda(x)$ згідно з (3.11). У лістингу 3.3 наведено програмну реалізацію отримання поліному $\lambda(x)$.

Лістинг 3.3. Програмний метод отримання поліному локаторів стирань $\lambda(x)$

```
private PolynomialBracket ObtainErasureLocatorsPolynomial()
{
    var lambdaPolynomial = new Polynomial()
    {
        NumberOfBrackets = numberOfErasures,
        polynomialBrackets = new List<PolynomialBracket>()
    };
```

```
    for (int i = 0; i < lambdaPolynomial.NumberOfBrackets; i++)
    {
        var constantTerm = new PolynomialTerm()
        {
            Power = 0,
            Coefficient = 1
        };
        var xTerm = new PolynomialMixedTerm()
        {
            Power = 1,
            TermCoefficient = new PolynomialTerm()
            {
                Power = erasureLocators[i],
                Coefficient = -1
            }
        };
        var polynomial = new PolynomialBracket()
        {
            alphaTerms = new List<PolynomialTerm> { constantTerm },
            xAlphaTerms = new List<PolynomialMixedTerm> { xTerm }
        };

        lambdaPolynomial.polynomialBrackets.Add(polynomial);
    }
    foreach (var term in lambdaPolynomial.polynomialBrackets)
    {
        var alpha = new List<PolynomialTerm>
{ term.xAlphaTerms.First().TermCoefficient };
        term.xAlphaTerms.First().TermCoefficient =
polynomialProcessing.TransformAlphaTerms(alpha).First();
    }
    var erasureLocatorsPolynomial = new PolynomialBracket();
    if (lambdaPolynomial.NumberOfBrackets > 1)
    {
        erasureLocatorsPolynomial = lambdaPolynomial.polynomialBrackets[0];
        for (int i = 1; i < lambdaPolynomial.NumberOfBrackets; i++)
        {
            var secondBracket = lambdaPolynomial.polynomialBrackets[i];
            erasureLocatorsPolynomial =
polynomialMultiplier.MultiplyingTwoBrackets(erasureLocatorsPolynomial, secondBracket);
        }
    }
    else
    {
        erasureLocatorsPolynomial = lambdaPolynomial.polynomialBrackets.First();
    }

    return erasureLocatorsPolynomial;
}
```

Тепер можна знайти максимальну кількість χ помилок, які можуть бути виправлені після виявлення δ стирань:

$$\chi = \left\lfloor \frac{r-\delta}{2} \right\rfloor. \tag{3.13}$$

При цьому, потрібно брати до уваги, що дійсна кількість γ помилок повинна задовольняти нерівність $0 \le \gamma \le \chi$.

Крок 2. На наступному етапі потрібно обчислити компоненти $S_1, S_2, \ldots, S_r$ синдрому спотворень:

$$S_i = c'(2^i), i = 1,2,\ldots,r,$$
$$c'(x) = \sum_{j=0}^{n-1} c'_j x_j \,, \tag{3.14}$$

де c'_j – розряди зчитаної послідовності ШК-знаків. У разі якщо всі компоненти синдрому дорівнюють нулю, вважається, що у зчитаній послідовності немає спотворень.

На основі компонентів $S_1, S_2, \ldots, S_r$ синдрому спотворень формується синдромний поліном вигляду:

$$S(x) = 1 + \sum_{i=1}^{r} S_i x^i \,. \tag{3.15}$$

У лістингу 3.4 наведено програмну реалізацію знаходження синдромного поліному.

Лістинг 3.4. Програмний метод знаходження синдромного поліному $S(x)$

```
private PolynomialBracket ObtainSyndromePolynomial()
{
    TranslatingBetweenPolynomialAndCodewords translatingBetweenPolynomialAndCodewords
= new TranslatingBetweenPolynomialAndCodewords(composingGF);
    var codePolynomial =
translatingBetweenPolynomialAndCodewords.ObtainPolynomialFromCodeword
       (encodedDataInPOmegaNotation);

    var syndromPolynomial = new PolynomialBracket()
    {
```

```
        alphaTerms = new List<PolynomialTerm>(),
        xAlphaTerms = new List<PolynomialMixedTerm>(),
        xTerms = null
    };

    var constantTerm = new PolynomialTerm() { Power = 0, Coefficient = 1 };
    syndromPolynomial.alphaTerms.Add(constantTerm);

    for (int i = 1; i <= encodingSettings.NumberOfControlCodewords; i++)
    {
        var alphaMultiplier = new PolynomialTerm()
        {
            Power = i,
            Coefficient = 1
        };

        var distortionSyndrome =
polynomialProcessing.ObtainPolynomialOfAlpha(codePolynomial, alphaMultiplier);
        var ternaryCodeOfDistortionSyndrom =
polynomialProcessing.ConvertingAlphaPolynomialToTernaryCode(distortionSyndrome,
composingGF.DigitCapacity);

        var xAlpha = new PolynomialMixedTerm()
        {
            Power = i,
            TermCoefficient = new PolynomialTerm()
            {
                Coefficient = 1,
                Power =
composingGF.GetGFElement(ternaryCodeOfDistortionSyndrom).AlphaPower
            }
        };

        syndromPolynomial.xAlphaTerms.Add(xAlpha);
    }

    return syndromPolynomial;
}
```

Крок 3. Тепер, знаючи синдромний поліном, зі співвідношення $S(x)\lambda(x) \equiv V(x) \bmod x^{r+1}$ можна отримати синдромний поліном спотворень $V(x)$:

$$(1+\sum_{w=1}^{r} S_w x^w)\lambda(x) = 1+\sum_{w=1}^{r} V_w x^w \bmod x^{r+1}, \qquad (3.16)$$

де $V_1, V_2, \ldots, V_r$ – компоненти модифікованого синдрому спотворень.

У лістингу 3.5 наведено програмну реалізацію знаходження синдромного поліному спотворень.

Лістинг 3.5. Програмний метод знаходження синдромного поліному спотворень $V(x)$

```
private PolynomialBracket CalculatePolynomialProductModX(PolynomialBracket
firstBracket, PolynomialBracket secondBracket)
{
    var polynomial = polynomialMultiplier.MultiplyingTwoBrackets(firstBracket,
secondBracket);
    polynomial =
polynomialProcessing.PolynomialAdditionInCommonMiltiplierToGetAlphaPolynomial
       (polynomial);
    bool oneMoreLoop = true;
    while (oneMoreLoop)
    {
        oneMoreLoop = false;
        for (int i = 0; i < polynomial.xAlphaTerms.Count; i++)
        {
            if (polynomial.xAlphaTerms[i].Power >=
(encodingSettings.NumberOfControlCodewords + 1) ||
                polynomial.xAlphaTerms[i].TermCoefficient.Power == int.MinValue)
            {
                polynomial.xAlphaTerms.RemoveAt(i);
                oneMoreLoop = true;
            }
        }
    }

    return polynomial;
}
```

Крок 4. На цьому етапі необхідно обчислити синдром помилок. Для цього з послідовності $V_1, V_2, \ldots, V_r$ виділяється підпослідовність $V_j, V_{j+1}, \ldots, V_r$, де $j = r - 2\chi + 1$. Виділена підпослідовність тепер подається у вигляді послідовності $D_1, D_2, \ldots, D_{2\chi}$, де $D_\varepsilon = V_{r-2\chi+\varepsilon}$, $\varepsilon = 1, 2, \ldots, 2\chi.$

Тоді $D_1, D_2, \ldots, D_{2\chi}$ є компонентами синдрому помилок. У лістингу 3.6 наведено програмну реалізацію обчислення цих компонентів.

Лістинг 3.6. Програмний метод обчислення компонентів D_ε синдрому помилок

```
private List<PolynomialTerm> ObtainComponentsOfErrorSyndrome(PolynomialBracket
modifiedSyndromePolynomial)
{
```

```
    CheckErrors();
    int epsilon = 2 * maxNumberOfErrors;

    List<PolynomialTerm> errorSyndromeComponent = new List<PolynomialTerm>();

    for (int i = 1; i <= epsilon; i++)
    {
        var component = GetAlphaByXPower
            (modifiedSyndromePolynomial, encodingSettings.NumberOfControlCodewords - 2
* maxNumberOfErrors + i);
        errorSyndromeComponent.Add(component);
    }

    return errorSyndromeComponent;
}
```

Крок 5. Наступним кроком необхідно знайти поліном локаторів помилок. Коефіцієнти σ_j полінома локаторів помилок $\sigma(x)=1+\sum_{j=1}^{\gamma}\sigma_j x^j$ можуть бути обчислені на основі компонент D_ε синдрому помилок матричним способом [128].

Для цього складається матричне рівняння наступного вигляду:

$$\begin{bmatrix} D_1 & D_2 & \cdots & D_\gamma \\ D_2 & D_3 & \cdots & D_{\gamma+1} \\ & & \cdots & \\ D_\gamma & D_{\gamma+1} & \cdots & D_{2\gamma-1} \end{bmatrix} \times \begin{bmatrix} \sigma_\gamma \\ \sigma_{\gamma-1} \\ \cdots \\ \sigma_1 \end{bmatrix} = -\begin{bmatrix} D_{\gamma+1} \\ D_{\gamma+2} \\ \cdots \\ D_{2\gamma} \end{bmatrix}. \tag{3.17}$$

Оскільки кількість стирань δ відома до початку процедури виправлення помилок, то кількість помилок спочатку приймається $\gamma=\chi=\left\lfloor\frac{r-\delta}{2}\right\rfloor$ і знаходиться визначник матриці M вигляду:

$$M=\begin{bmatrix} D_1 & \cdots & D_\gamma \\ & \cdots & \\ D_\gamma & \cdots & D_{2\gamma-1} \end{bmatrix}. \tag{3.18}$$

Якщо визначник не дорівнює нулю, то обране значення γ є правильним.

Тоді можна скласти матричне рівняння:

$$\begin{bmatrix} \sigma_\gamma \\ \cdots \\ \sigma_1 \end{bmatrix} = M^{-1} \times \begin{bmatrix} -D_{\gamma+1} \\ \cdots \\ -D_{2\gamma} \end{bmatrix}. \tag{3.19}$$

Якщо ж визначник матриці M дорівнює нулю, то значення γ має бути зменшене на одиницю, після чого процедура повторюється. Значення γ зменшується доти, доки не буде отримано визначник, відмінний від нуля.

У лістингу 3.7 наведено програмну реалізацію знаходження поліному локаторів помилок $\sigma(x)$.

Лістинг 3.7. Програмний метод отримання поліному локаторів помилок $\sigma(x)$

```
private PolynomialBracket ObtainErrorLocatorsPolynomial(List<PolynomialTerm>
componentsOfErrorSyndrome)
{
    numberOfErrors = maxNumberOfErrors;
    MatrixDeterminant matrixDeterminant = new MatrixDeterminant(composingGF);
    ServingMatrixMethods matrixMethods = new ServingMatrixMethods(composingGF);
    MatrixMultiplication matrixMultiplication = new MatrixMultiplication(composingGF);

    int[,] matrix;

    while (true)
    {
        matrix = new int[numberOfErrors, numberOfErrors];
        for (int i = 0, k = 0; i < matrix.GetLength(0); i++, k++)
        {
            for (int j = 0, q = k; j < matrix.GetLength(1); j++, q++)
            {
                matrix[i, j] = componentsOfErrorSyndrome[q].Power;
            }
        }

        matrixDeterminant.GetMatrixDeterminant(matrix);
```

```
        if (matrixDeterminant.Determinant >= 0) break;
        numberOfErrors--;
    }

    var inversedMatrix = matrixMethods.ObtainInverseMatrix(matrix);
    var errorSyndromComponentsVector =
GetComponentsOfErrorSyndromVector(componentsOfErrorSyndrome);

    var errorLocators = matrixMultiplication.MultiplyMatrixByVector(inversedMatrix,
errorSyndromComponentsVector);

    var errorLocatorsPolynomial = new PolynomialBracket()
    {
        alphaTerms = new List<PolynomialTerm> { new PolynomialTerm { Coefficient = 1,
Power = 0 } },
        xAlphaTerms = new List<PolynomialMixedTerm>(),
        xTerms = null
    };

    int count = 1;
    foreach (var item in errorLocators)
    {
        var term = new PolynomialMixedTerm
        {
            Power = count,
            TermCoefficient = new PolynomialTerm { Coefficient = 1, Power = item }
        };
        errorLocatorsPolynomial.xAlphaTerms.Add(term);
    }

    return errorLocatorsPolynomial;
}
```

Крок 6. Рівняння $\sigma(x) = 0$ має бути розв'язане в полі $GF(P_\Omega)$. Корені рівняння дорівнюватимуть. Тоді локатори помилок $X_{p_1}, X_{p_2}, \ldots, X_{p_\gamma}$ дорівнюють: $X_{p_1} = x_1^{-1}$, $X_{p_2} = x_2^{-1}$, ... , $X_{p_\gamma} = x_\gamma^{-1}$, де x_1, x_2, ... , x_γ – корені рівняння $\sigma(x) = 0$.

У лістингу 3.8 наведено програмну реалізацію знаходження локаторів помилок, тобто розрядів послідовності ШК-знаків, в яких містяться помилки.

Лістинг 3.8. Програмний метод знаходження локаторів помилок

```
private List<int> ObtainErrorLocators(PolynomialBracket errorLocatorsPolynomial)
{
    var errorLocators = new List<int>();
    SolveEquation solveEquation = new SolveEquation();

    if (errorLocatorsPolynomial.xAlphaTerms.Count == 1)
```

```
    {
        var solution = solveEquation.LinearEquation(errorLocatorsPolynomial);
        errorLocators.Add(solution * (-1));
    }

    return errorLocators;
}
```

Крок 7. Поліном значень спотворень $Q(x)$ степеню r можна знайти зі співвідношення: $V(x)\sigma(x) \equiv Q(x) \bmod x^{r+1}$. Для знаходження поліному скористаємось методом, наведеним у лістингу 3.5.

Крок 8. Формується спільна множина X локаторів спотворень, що містить локатори стирань та локатори помилок: $X = \{X_1, X_2, \ldots, X_{\delta+\gamma}\}$.

Крок 9. Тепер можна обчислити величини E_β спотворень, причому як стирань, так й помилок:

$$E_\beta = \frac{Q(X_\beta^{-1})}{\prod\limits_{\substack{\beta=1,\, j=1 \\ \beta \neq j}} (1 - X_\beta^{-1} X_j)}. \tag{3.20}$$

У лістингу 3.9 наведено програмну реалізацію обчислення величин спотворень.

Лістинг 3.9. Програмний метод обчислення величин E_β спотворень

```
private List<int> ObtainDistortionValues(PolynomialBracket distortionValuesPolynomial,
List<int> distortionLocators)
{
    var distortionValues = new List<int>();
    for (int beta = 0; beta < distortionLocators.Count; beta++)
    {
        var alphaTerm = new PolynomialTerm
        {
            Coefficient = 1,
            Power = distortionLocators[beta] * (-1)
        };
        var q =
polynomialProcessing.ObtainPolynomialOfAlpha(distortionValuesPolynomial, alphaTerm);
        var numenator = polynomialProcessing.SumUpAlphaTerms(q);
        var product = new Polynomial { NumberOfBrackets = 0, polynomialBrackets = new
List<PolynomialBracket>() };
        for (int j = 0; j < distortionLocators.Count; j++)
        {
            if (beta != j)
            {
                var bracket = new PolynomialBracket
                {
                    xTerms = null,
                    xAlphaTerms = null,
                    alphaTerms = new List<PolynomialTerm> { new PolynomialTerm
{ Coefficient = 1, Power = 0 } }
                };
                var term1 = distortionLocators[beta] * (-1);
                var term2 = distortionLocators[j];
                var term = new PolynomialTerm
                {
                    Coefficient = 1,
                    Power = alphaPowerCalculation.MultiplyAlphaTermWithCoefficient
                            (alphaPowerCalculation.MultiplyAlphaTerms
                                                        (term1, term2), 2)
                };
                bracket.alphaTerms.Add(term);
                product.polynomialBrackets.Add(bracket);
                product.NumberOfBrackets++;
            }
        }
        var firstBracket = product.polynomialBrackets[0];
        if (product.NumberOfBrackets > 1)
        {
            for (int i = 1; i < product.NumberOfBrackets; i++)
            {
                var secondBracket = product.polynomialBrackets[i];
                firstBracket =
polynomialMultiplier.MultiplyingTwoBrackets(firstBracket, secondBracket);
            }
        }
        var denominator =
polynomialProcessing.SumUpAlphaTerms(firstBracket.alphaTerms);

        distortionValues.Add(alphaPowerCalculation.DivideAlphaTerms(numenator.Power,
denominator.Power));
    }

    return distortionValues;
}
```

Крок 10. Останнім етапом виправляються наявні спотворення. Для цього для кожної пари $\langle X_\beta, E_\beta \rangle$, де $\beta = 1, 2, \ldots, \delta + \gamma$, X_β записуються у вигляді $X_\beta = \alpha^f$ та знаходиться f. Тоді у f-му розряді зчитаної послідовності ШК-знаків може бути виконана корекція: $c_f = (c'_f - E_\gamma) \bmod m_s(x)$.

Якщо X_β – локатори стирань, то E_β – різниці між вибраними значеннями стертих символів з локатором X_β та істинними символами. Якщо X_β – локатори помилок, то E_β – різниці між отриманими (зчитаними) символами з локаторами X_β та істинними символами.

В результаті отримуємо декодовану послідовність ШК-знаків, які тепер можна перетворити на початкові алфавітно-цифрові дані.

3.3. Дослідження процедури забезпечення завадостійкості штрихових кодів з трьома градаціями кольору

Дослідимо описану у п. 3.2.4 процедуру декодування даних. Для цього візьмемо закодовану в п. 3.2.3 послідовність 3 638 22 716 58 679 29 173 157, що відповідає текстовому рядку *Sikorsky*.

Нехай в результаті процедури розпізнавання ДШК-позначки зчитана послідовність $n = 9$ ШК-знаків 3 638 22 –1 58 679 29 100 157. Запишемо її у наступному вигляді, де α^k – це k-тий розряд послідовності, $k = 0, 1, \ldots, n$.

X_i:	α^8	α^7	α^6	α^5	α^4	α^3	α^2	α^1	α^0
C':	3	638	22	**–1**	58	679	29	100	157

Вважатимемо, що у п’ятому розряді має місце стирання (цей розряд виділено жирним), яке в результаті розпізнавання було замінене на значення -1, а у першому розряді виникла помилка (виділено підкресленням).

Тоді зчитаній послідовності відповідає наступний кодовий поліном:

$$c'(x) = 3x^8 + 638x^7 + 22x^6 - x^5 + 58x^4 + 679x^3 + 29x^2 + 100x + 157.$$

Декодувати зчитану послідовність ШК-знаків будемо згідно з кроками процедури, описаної в п. 3.2.4.

Спочатку зафіксуємо номери $i_1, i_2, ..., i_\delta$ розрядів, які містять стирання, де δ – кількість стирань. Для прикладу, що досліджується, ці параметри мають значення $i_1 = 5$ та $\delta = 1$. Перш ніж переходити до подальших дій, необхідно перевірити, чи коректувальний код з обраним рівнем корекції спотворень $\Lambda = 2$. Оскільки за такого рівня $r = 3$, нерівність $\delta \le r$ виконується, отже коректувальний код може виправити таку кількість стирань.

Знаючи розряди, в яких мають місце стирання (у нашому випадку – це один, п’ятий розряд), можемо визначити локатори стирань: $X_{i_1} = \alpha^5$. Згідно з поданням α^5 у десятковому вигляді маємо $X_{i_1} = 243$.

Побудуємо тепер за формулою (3.11) поліном локаторів стирань:

$$\lambda(x) = 1 - 243x = 1 - \alpha^5 x = 1 + 2\alpha^5 x = 1 + \alpha^{369} x = 1 + 486x.$$

З формули (3.13) можемо знайти максимальну кількість χ помилок, яка може бути виправлена коректувальним кодом після виявлення δ стирань:

$$\chi = \left\lfloor \frac{r-\delta}{2} \right\rfloor = \left\lfloor \frac{3-2}{2} \right\rfloor = 1.$$

Отже, у досліджуваному прикладі може бути виправлена одна помилка.

Обчислимо за формулою (3.14) компоненти $S_1, S_2, \ldots, S_r$ синдрому спотворень:

$$S_1 = c'(\alpha^1) = \alpha^1 \cdot \alpha^8 + \alpha^{497} \cdot \alpha^7 + \alpha^{148} \cdot \alpha^6 + \alpha^{400} \cdot \alpha^4 + \alpha^{95} \cdot \alpha^3 + \alpha^{382} \cdot \alpha^2 +$$
$$+ \alpha^{158} \cdot \alpha^1 + \alpha^{648} = \alpha^9 + \alpha^{504} + \alpha^{154} + \alpha^{404} + \alpha^{98} + \alpha^{384} + \alpha^{159} + \alpha^{648} =$$
$$= \alpha^{501} = 535.$$

Таблиця 3.1. Порозрядна трійкова сума доданків компоненту S_1 синдрому спотворень

α^9	0 2 1 0 0 0
α^{504}	2 1 2 2 2 1
α^{154}	0 0 1 1 0 1
α^{404}	1 1 0 1 2 0
α^{98}	0 1 2 0 1 1
α^{384}	1 0 0 2 0 0
α^{159}	1 0 2 0 1 0
α^{648}	0 1 2 2 1 1
$\alpha^{501} = 535$	2 0 1 2 1 1

$$S_2 = c'(\alpha^2) = \alpha^1 \cdot \alpha^{16} + \alpha^{497} \cdot \alpha^{14} + \alpha^{148} \cdot \alpha^{12} + \alpha^{400} \cdot \alpha^8 + \alpha^{95} \cdot \alpha^6 + \alpha^{382} \cdot \alpha^4 +$$
$$+ \alpha^{158} \cdot \alpha^2 + \alpha^{648} = \alpha^{17} + \alpha^{511} + \alpha^{160} + \alpha^{408} + \alpha^{101} + \alpha^{386} + \alpha^{160} + \alpha^{648} =$$
$$= \alpha^{253} = 593.$$

Таблиця 3.2. Порозрядна трійкова сума доданків компоненту S_2 синдрому спотворень

α^{17}	1 0 0 2 0 1
α^{511}	2 0 0 0 1 1
α^{160}	0 2 0 1 2 1
α^{408}	2 2 0 1 2 1
α^{101}	0 1 1 2 2 2
α^{386}	0 2 0 2 1 0
α^{160}	0 2 0 1 2 1
α^{648}	0 1 2 2 1 1
$\alpha^{253} = 593$	2 1 0 2 2 2

$$S_3 = c'(\alpha^3) = \alpha^1 \cdot \alpha^{24} + \alpha^{497} \cdot \alpha^{21} + \alpha^{148} \cdot \alpha^{18} + \alpha^{400} \cdot \alpha^{12} + \alpha^{95} \cdot \alpha^9 + \alpha^{382} \cdot \alpha^6 +$$
$$+ \alpha^{158} \cdot \alpha^3 + \alpha^{648} = \alpha^{25} + \alpha^{518} + \alpha^{166} + \alpha^{412} + \alpha^{104} + \alpha^{388} + \alpha^{161} + \alpha^{648} =$$
$$= \alpha^{205} = 698.$$

Таблиця 3.3. Порозрядна трійкова сума доданків компоненту S_3 синдрому спотворень

α^{25}	1 2 0 2 1 0
α^{518}	0 0 2 2 0 2
α^{166}	1 2 2 2 1 1
α^{412}	2 2 0 2 2 1
α^{104}	2 2 2 2 0 1
α^{388}	0 2 1 0 1 2
α^{161}	2 0 1 2 1 0
α^{648}	0 1 2 2 1 1
$\alpha^{205} = 698$	2 2 1 2 1 2

Маючи компоненти синдрому спотворень, можемо утворити з (3.15) синдромний поліном:

$$S(x) = 1 + \alpha^{501}x + \alpha^{253}x^2 + \alpha^{205}x^3 = 1 + 535x + 593x^2 + 698x^3.$$

Тепер, використовуючи синдромний поліном та поліном локаторів стирань, можемо знайти зі співвідношення (3.16) модифікований синдромний поліном спотворень $V(x)$:

$$V(x) = S(x)\lambda(x) \bmod x^{r+1} = (1 + \alpha^{501}x + \alpha^{253}x^2 + \alpha^{205}x^3) \cdot (1 + \alpha^{369}x) \bmod x^4 =$$
$$= 1 + \alpha^{643}x + \alpha^{591}x^2 + \alpha^{592}x^3 = 1 + 292x + 193x^2 + 579x^3.$$

Знайдемо синдром помилок. Для цього, згідно з кроком 4, з послідовності $V_1, V_2, \ldots, V_r$ виділяється підпослідовність $D_1, D_2, \ldots, D_{2\chi}$, де $D_\varepsilon = V_{r-2\chi+\varepsilon}$, $\varepsilon = 1, 2, \ldots, 2\chi$. Оскільки для досліджуваного прикладу $\chi = 1$, то $\varepsilon = \{1; 2\}$, відповідно:

$$D_1 = V_{3-2\cdot1+1} = V_2 = \alpha^{591} = 193,$$
$$D_2 = V_{3-2\cdot1+2} = V_3 = \alpha^{592} = 579.$$

Наступним кроком шукаємо поліном локаторів помилок $\sigma(x)$. Коефіцієнти σ_j поліному обчислюються на основі компонентів D_ε синдрому помилок матричним методом.

Для досліджуваного прикладу $\gamma = 1$, відповідно, згідно з формулою (3.18) матриця M складатиметься з одного елемента:

$$M = D_1 = \left[\alpha^{591}\right] = 193\,.$$

Визначник матриці дорівнює $\det M = 193$, а в зчитаній послідовності ШК-знаків міститься лише одна помилка.

Матричне рівняння (3.19) для досліджуваного прикладу матиме вигляд:

$$\sigma_1 = M^{-1}\left[-D_{2\gamma}\right].$$

Для його розв'язання спочатку обчислимо обернену матрицю M^{-1}. Враховуючи, що матриця M складається з одного елементу, для знаходження оберненої достатньо знайти обернене до цього елементу значення:

$$(193)^{-1} = (\alpha^{591})^{-1} = \alpha^{-591} = \alpha^{137} = 314.$$

Тоді матричне рівняння набуває вигляду:

$$\sigma_1 = \left[\alpha^{137}\right] \times \left[-\alpha^{592}\right] = \left[\alpha^{137}\right] \times \left[2\alpha^{592}\right] = 2\alpha^1 = \alpha^{364} \cdot \alpha^1 = \alpha^{365} = 6.$$

Отже, поліном локаторів помилок має вигляд:

$$\sigma(x) = 1 + \alpha^{365} x\,.$$

Згідно з кроком 6, формуємо тепер рівняння $\sigma(x) = 0$ та розв'язуємо його у полі $GF(3^6)$:

$$1+\alpha^{365}x=0.$$

Отримуємо корінь рівняння $x_1=\frac{-1}{\alpha^{365}}=\frac{2}{\alpha^{365}}=\frac{\alpha^{364}}{\alpha^{365}}=\alpha^{-1}$.

Тоді локатор помилки дорівнює $X_{p_1}=x_1^{-1}=\alpha^1$, а отже – помилка міститься у першому розряді зчитаної послідовності.

Знаходимо тепер поліном значень спотворень $Q(x)$ степеню r:

$$Q(x)=V(x)\sigma(x) \bmod x^4=(1+\alpha^{643}x+\alpha^{591}x^2+\alpha^{592}x^3)\cdot(1+\alpha^{365}) \bmod x^4=$$
$$=1+\alpha^{53}x+\alpha^{250}x^2=1+289x+156x^2.$$

Формуємо спільну множину X, яка містить локатори стирань та локатори помилок: $X=\ \alpha^5;\alpha^1\ $.

Відповідно до кроку 9 можемо обчислити тепер величини E_β спотворень – стирань і помилок – за формулою (3.20):

$$E_1=\frac{Q(X_1^{-1})}{1-X_1^{-1}X_2}=\frac{1+\alpha^{53}\cdot\alpha^{-5}+\alpha^{250}\cdot\alpha^{-10}}{1-\alpha^{-5}\cdot\alpha^1}=\frac{1+\alpha^{48}+\alpha^{240}}{1+2\alpha^{-4}}=$$
$$=\frac{1+\alpha^{48}+\alpha^{240}}{1+\alpha^{364}\cdot\alpha^{724}}=\frac{\alpha^{392}}{\alpha^{318}}=\alpha^{74};$$
$$E_2=\frac{Q(X_2^{-1})}{1-X_2^{-1}X_1}=\frac{1+\alpha^{53}\cdot\alpha^{-1}+\alpha^{250}\cdot\alpha^{-2}}{1-\alpha^{-1}\cdot\alpha^5}=\frac{1+\alpha^{52}+\alpha^{248}}{1+2\alpha^4}=$$
$$=\frac{1+\alpha^{52}+\alpha^{248}}{1+\alpha^{364}\cdot\alpha^4}=\frac{\alpha^{589}}{\alpha^{686}}=\alpha^{631}.$$

Останнім кроком маємо виправити спотворення. Для цього виконується корекція згідно з кроком 10:

$$c_1' = c_1 - E_1 = 0 - \alpha^{74} = 2\alpha^{74} = \alpha^{364} \cdot \alpha^{74} = \alpha^{438} = 716;$$
$$c_2' = c_2 - E_2 = \alpha^{158} - \alpha^{631} = \alpha^{533} = 415.$$

Після корекції стирання у п’ятому та помилки у першому розрядах послідовність ШК-знаків набуває вигляду: 3 628 22 716 58 679 29 415 157.

Для отримання початкових даних виправлений кодовий поліном:

$$c(x) = 3x^8 + 628x^7 + 22x^6 + 716x^5 + 58x^4 + 679x^3 + 29x^2 + 415x + 157$$

потрібно поділити на твірний поліном (3.10). Після здійснення операції ділення отримуємо послідовність інформаційних ШК-знаків: 403 470 262 436 720 302, яке не містить помилок та відповідає текстовому рядку *Sikorsky*.

3.4. Аналіз можливостей коректувального коду для завадостійкості штрихових кодів з трьома градаціями кольору

Розглянемо можливості коректувального коду, який здатний виправляти спотворення двох типів – помилок та стирань, забезпечуючи цим завадостійкість ДШК-позначок у разі ураження певної частини графічного зображення позначки, нанесеного на носій.

З нерівностей (3.2) та (3.3) можемо одержати наступні співвідношення:

$$r = 2^{\Lambda} - 1 \text{ або } r + 1 = 2^{\Lambda}. \qquad (3.21)$$

З цих співвідношень логічно випливає, що кількість контрольних ШК-знаків r має бути непарною.

Тоді за заданого рівня корекції спотворень Λ й, відповідно, за відомого значення r допустима кількість ушкоджених ШК-знаків, за якої коректна обробка ДШК-позначки є можливою, знаходиться в діапазоні від $\frac{r+1}{2}$ (що відповідає виправленню $\frac{r-1}{2}$ помилок та одного стирання) до r (що відповідає виправленню r стирань та жодної помилки).

Тоді усереднена допустима кількість F_c ушкоджених ШК-знаків у ДШК-позначці за заданого r становитиме:

$$F_c = \left\lfloor \frac{r+1}{2} + 2 \right\rfloor = \left\lfloor \frac{3r+1}{4} \right\rfloor.$$

Наприклад, за $r = 7$ (табл. 3.4) допустима кількість ушкоджених ШК-знаків, за яких досі можливе правильне оброблення ДШК-позначки, знаходиться в діапазоні 4-7, тоді як усереднена допустима кількість ушкоджених ШК-знаків становить 5 ШК-знаків.

Таблиця 3.4. Допустима кількість ушкоджених ШК-знаків у ДШК-позначці за $r = 7$

Кількість додаткових (контрольних) кодовекторів, r	Допустима кількість ушкоджених ШК-знаків		
	Загалом	Помилкових ШК-знаків	Стертих ШК-знаків
7	4	3	1
	5	2	3
	6	1	5
	7	-	7

Якщо під час використання ДШК-позначки у ШК-знаках здебільшого виникають помилки, стирання ж є малоймовірними, то допустима кількість ушкоджених ШК-знаків у позначці за заданого r

становитиме $F_a = \dfrac{r-1}{2}$, де F_a є нижньою границею значення усередненої допустимою кількості.

Тоді допустиму площу ураження ДШК-позначки, за якої вона досі має правильно оброблюватись, можна визначити наступним чином:

$$S_c(k,r) = \frac{F_c}{k+r} = \frac{\lceil (3r+1)/4 \rceil}{k+r},$$
$$S_a(k,r) = \frac{F_a}{k+r} = \frac{(r-1)/2}{k+r} = \frac{r-1}{2(k+r)},$$

де k – кількість інформаційних ШК-знаків,

S_c – усереднене значення,

S_a – нижня границя значення.

Залежність допустимої прощі ураження ДШК-позначки від кількості контрольних ШК-знаків показана у табл. 3.8.

Надлишковість ДШК-позначки можна визначити з відношення кількості контрольних ШК-знаків до інформаційної ємності ДШК позначки, тобто кількості інформаційних ШК-знаків:

$$R = \frac{r}{k}.$$

Тоді за достатньо великих значень k та r формули S_c та S_a можна визначити як функції від R:

$$S_c(R) \approx \frac{\lceil 3r/4 \rceil}{k+r} = \frac{3}{4} \cdot \frac{R}{1+R},$$
$$S_a(R) = \frac{r}{2(k+r)} = \frac{1}{2} \cdot \frac{R}{1+R}.$$

Функції $S_c(R)$ та $S_a(R)$ показують залежність допустимої площі ураження ДШК-позначки, за якого ще можливе правильне розпізнавання позначки, від коефіцієнта надлишковості R та кількості контрольних ШК-знаків r.

Наприклад, якщо кількість контрольних ШК-знаків однакова із кількістю інформаційних ШК-знаків ($k = r$, що фактично є дублюванням ДШК-позначки), то допустима площа ураження штрихкодового зображення становить 25-37% (рис. 3.1), де 25% − нижня границя значення, за якого при ураженні виникають лише помилки, а 37% − усереднене значення, коли, окрім помилок, виникають також стирання.

Таблиця 3.5. Допустима площа ураження ДШК-позначки

Кількість контрольних розрядів, r	Допустима площа ураження ДШК-позначки
0,2k	8,3% − 12,5%
0,3k	10,4% − 15,6%
0,4k	12,5% − 18,7%
0,5k	14,6% − 21,8%
0,6k	16,7% − 24,9%
0,7k	18,8% − 28%
0,8k	20,9% − 31,2%
0,9k	23% − 34,3%
k	25% − 37,5%

У п. 3.2.2. йшлося про те, що максимальний рівень корекції спотворень доцільно визначати зі співвідношення $2^{\Lambda} - 1 = \frac{2}{3}k$, тобто коли

кількість r контрольних ШК-знаків становить дві третини від кількості k інформаційних ШК-знакі. За $r = 0,67k$ допустима площа ураження ДШК-позначки складатиме 20-30% (рис. 3.2).

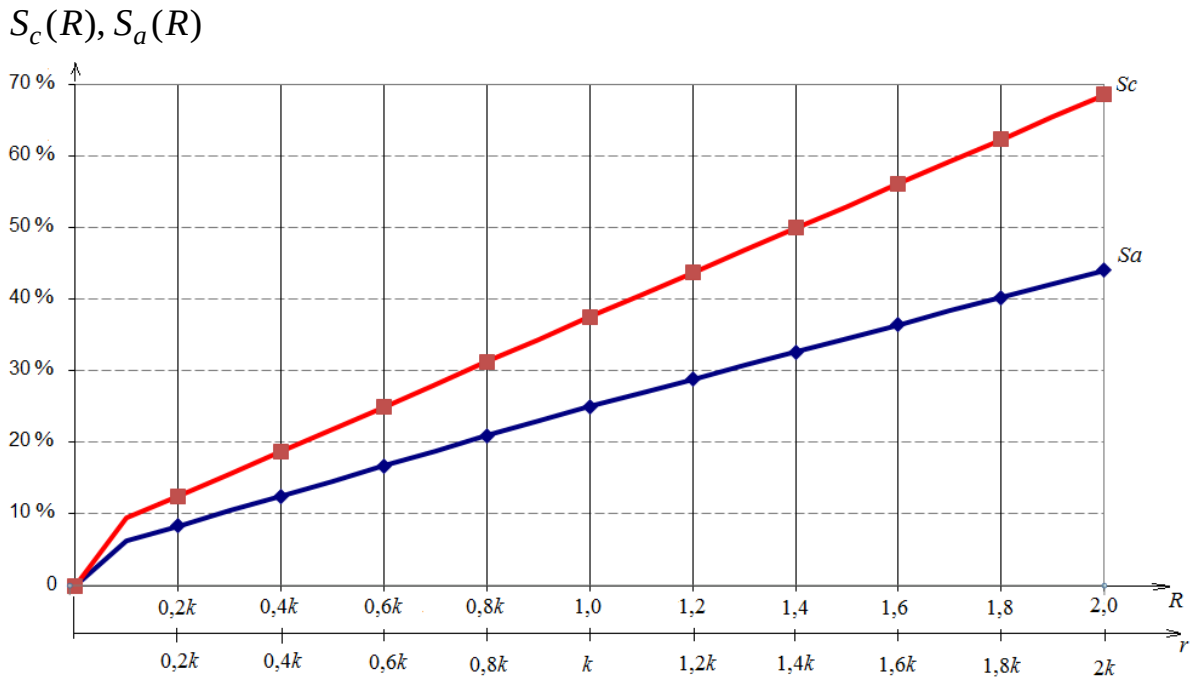
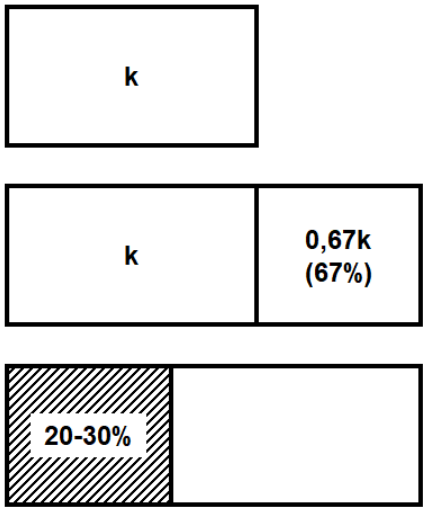


Рис. 3.1. Залежність допустимої площі S ураження ДШК-позначки від коефіцієнта надлишковості R



Кількість інформаційних ШК-знаків (інформаційна ємність ДШК-позначки)

Введення надлишковості (контрольних ШК-знаків)

Завадозахищеність (допустима площа ураження – 20-30%)

Рис. 3.2. Аналіз завадостійкості ДШК-позначки за $r = \frac{2}{3}k$

Таким чином, розробник програмної системи формування ДШК-позначок, беручи до уваги особливості галузі застосування їхніх графічних зображень, має можливість забезпечувати завадостійкість ДШК-позначок внаслідок обрання відповідної кількості контрольних ШК-знаків під час створення ДШК-позначки.

3.5. Висновки до третього розділу

Дослідження процедур забезпечення завадостійкого кодування даних, поданих штриховими кодами з трьома градаціями кольору, дозволяє зробити наступні висновки.

1. Визначено, що забезпечення завадостійкості інформації, яка зберігається у вигляді штрихового коду, підвищує надійність зчитування ДШК-позначок, а отже, розширює сферу застосування технології кодування інформації на основі штрихових кодів з трьома градаціями кольору.
2. Визначено, що для надійного зберігання даних у вигляді двовимірного штрихового коду під час кодування даних необхідно застосовувати коректувальний код Ріда-Соломона, який дозволяє виправляти багатократні спотворення многозначних символів – ШК-знаків.
3. Досліджено систему контроля спотворень та коректувальні можливості коду Ріда-Соломона, що дало змогу ввести п’ять рівнів корекції спотворень. Використання певного рівня залежить від низки факторів у конкретній галузі застосування штрихових кодів та має визначатись експертом під час проєктування програмної системи формування ДШК-позначок.
4. Визначено залежність ступеня завадозахищеності від кількості контрольних ШК-знаків.

5. Доведено, що запропонований алгоритмічно-програмний підхід до корекції можливих спотворень у ДШК-позначках забезпечує високий рівень завадостійкості даних, поданих у вигляді штрихового коду з трьома градаціями кольору.

РОЗДІЛ 4. ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРОЦЕСІВ АВТОМАТИЧНОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ ЛОГІСТИКИ НА ОСНОВІ ШТРИХОВИХ КОДІВ З ТРЬОМА ГРАДАЦІЯМИ КОЛЬОРУ

4.1. Вимоги до програмної системи для автоматичної ідентифікації об'єктів логістики на основі штрихових кодів з трьома градаціями кольору

Для розроблення базової архітектури програмної системи для автоматичної ідентифікації об'єктів логістики на основі штрихових кодів з трьома градаціями кольору потрібно визначити функціональні та нефункціональні вимоги до такої системи [5, 8, 13, 15, 17, 18, 20, 22–25, 28, 33, 36, 38, 39, 48, 49, 59, 65, 66, 68, 87, 100]. Функціональні вимоги визначають базову поведінку програмної системи та надають опис характеристик і функцій системи, що мають бути реалізовані, щоб кінцеві користувачі могли здійснювати ефективну взаємодію зі системою відповідно до її призначення. Нефункціональні вимоги визначають атрибути програмної системи, такі як безпека, надійність, швидкодія, масштабованість, зручність у використанні тощо.

Для формулювання вимог до програмної системи для автоматичної ідентифікації об'єктів логістики на основі штрихових кодів з трьома градаціями кольору потрібно визначити процеси, що мають підтримуватись та забезпечуватись цією програмною системою. При проєктуванні системи автоматичної ідентифікації об'єктів на основі технології штрихового кодування також потрібно враховувати особливості оброблення інформації, поданої у вигляді ШК-позначки.

На рис. 4.1 показані модулі логістичної системи, які стосуються роботи з даними про одиниці обліку, що можуть бути представлені у

вигляді ДДШК-позначок та ДДШК-документів.

У програмній системі для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору передбачається подання даних двох типів. Дані першого типу являють собою докладний текстовий опис об’єкта обліку, який утворює «автономну базу даних» про об’єкт. Максимальний обсяг даних такого типу може досягати 7 Кбайт. Дані другого типу являють собою посилання на відповідний запис у хмарному сховищі даних програмної системи. В цьому хмарному сховищі можуть зберігатись сукупності різнорідних файлів, які мають стосунок до об’єкта обліку: текстові документи, фотографії, відеозаписи тощо.

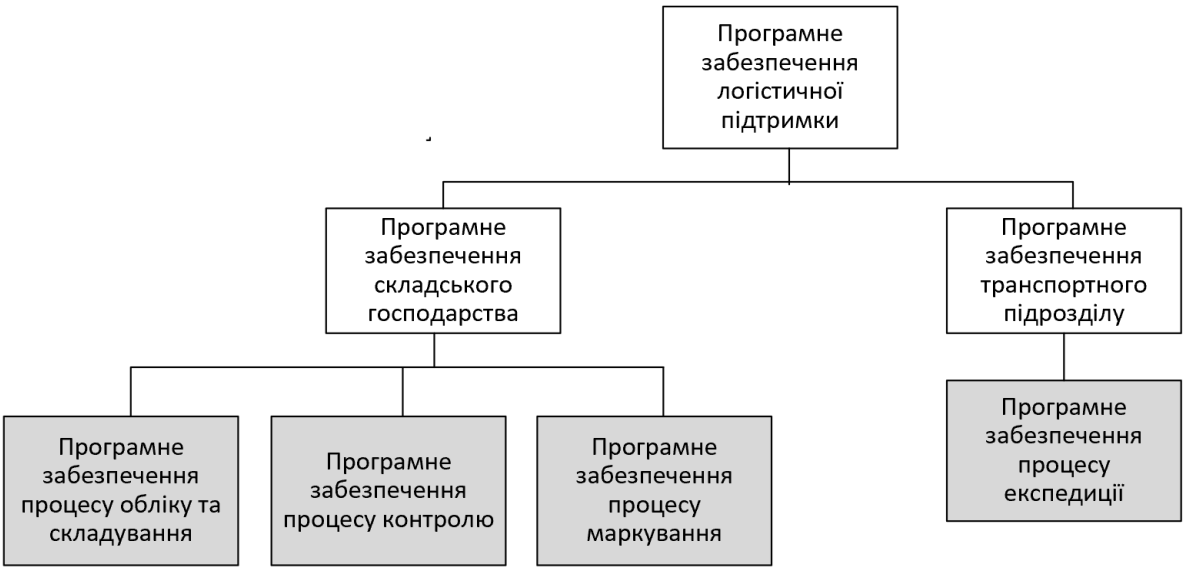


Рис. 4.1. Місце штрихового кодування у структурі програмного забезпечення логістичної підтримки (сірий колір)

Таке дворівневе подання даних може бути забезпечене або шляхом створення ШК-позначок двох різних типів відповідно до їхнього призначення, або за допомогою ДДШК, особливістю якого є можливість подання даних двох вищезазначених типів в одній позначці. Розглянемо другий випадок докладніше.

Основна інформація про об’єкт зберігається у штрихкодовій позначці, що знаходиться на самому об’єкті обліку. Інформація цього типу подається за допомогою нижнього рівня дворівневого штрихового коду. До цієї інформації завжди можна отримати доступ за допомогою портативного пристрою (смартфона, планшета з відповідним програмним забезпеченням або портативного сканера штрихових кодів). Повна інформація про об’єкт обліку міститься у хмарному сховищі. Доступ до цієї інформації можливий лише за умови підключення до мережі Інтернет та здійснюється за допомогою посилання, закодованого на верхньому рівні дворівневого штрихового коду.

Сформулюємо функціональні та нефункціональні вимоги до програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору з урахуванням пріоритетності кожної вимоги. При визначенні пріоритетності вимог будемо керуватись шкалою, яка є схожою до шкали, що використовується у методі MoSCoW [91], який передбачає чотири пріоритети вимог:

- «Must have» (система повинна задовольняти цій вимозі);
- «Should have» (бажано, щоб система задовольняла цій вимозі);
- «Could have» (система може задовольняти цій вимозі);
- «Won’t have this time» (система може не задовольняти цій вимозі).

Оскільки будемо визначати лише *базові* вимоги до програмної системи для автоматичної ідентифікації об’єктів логістики, які стосуються суто застосування технології штрихового кодування, а саме – штрихових кодів з трьома градаціями кольору, то у цій дисертаційній роботі пропонується використовувати дещо інакший підхід до визначення пріоритетів вимог.

Пропонується поділяти вимоги (як функціональні, так і нефункціональні) на два класи:

- незалежні вимоги,

- залежні вимоги.

Залежна вимога відрізняється від *незалежної вимоги* тим, що набуває сенсу лише за умови задовільнення іншої вимогі, від якої вона залежить змістовно.

Тоді визначимо наступну шкалу для пріоритизації вимог до програмної системи:

- «Обов'язково»,
- «Обов'язково за умови»,
- «Бажано»,
- «Можливо».

Пріоритет «Обов'язково» є аналогом пріоритету «Must have» та може бути визначений як для незалежної, так й для залежної вимоги.

Пріоритет «Обов'язково за умови» пропонується застосовувати для випадків, коли визначають *обов'язкову залежну* вимогу. Наприклад, вимога «Забезпечення вибору типу ручного сканера штрихових кодів» є залежною від вимоги «Забезпечення сканування штрихкодованої інформації». Якщо основна вимога не виконується (не є обов'язковою), то втрачається сенс у залежній вимозі.

Пріоритет «Бажано» є аналогом пріоритету «Should have» та може бути визначений як для незалежної, так й для залежної вимоги.

Пріоритет «Можливо» є аналогом пріоритету «Could have» та може бути визначений як для незалежної, так й для залежної вимоги.

При визначенні базових вимог до програмної системи недоцільно застосовувати пріоритет «Won't have this time», тому у запропонованій шкалі немає аналогу для цього пріоритету, визначеного у методі MoSCoW [91].

Таким чином, функціональні вимоги до програмної системи для автоматичної ідентифікації об'єктів логістики на основі штрихових кодів з трьома градаціями кольору визначимо, як показано у табл. 4.1.

Таблиця 4.1. Базові функціональні вимоги, що пов’язані із застосуванням технології штрихового кодування

Код	Вимога	Пріоритет
ФВ1	Забезпечення введення алфавітно-цифрової логістичної інформації та формування запису про об’єкт обліку	Обов’язково
ФВ2	Забезпечення формування та оброблення логістичних документів	Обов’язково
ФВ3	Забезпечення зберігання та доступу до даних про об’єкт обліку в сховищі даних	Обов’язково
ФВ4	Забезпечення аналізу та ущільнення вхідних алфавітно-цифрових даних про об’єкт обліку	Обов’язково
ФВ5	Забезпечення завадостійкого кодування ущільнених даних про об’єкт обліку	Обов’язково
ФВ6	Забезпечення формування штрихкодової позначки для ідентифікації об’єкта обліку на основі штрихових кодів з трьома градаціями кольору	Обов’язково
ФВ7	Забезпечення створення штрихкодованих логістичних документів на основі штрихових кодів з трьома градаціями кольору	Обов’язково
ФВ8	Забезпечення розпізнавання даних на позначках та документах, поданих у вигляді штрихового коду з трьома градаціями кольору	Обов’язково
ФВ9	Забезпечення декодування даних, поданих у вигляді штрихового коду з трьома градаціями кольору	Обов’язково
ФВ10	Забезпечення цільового оброблення декодованих даних	Обов’язково
ФВ11	Забезпечення інтеграції та обміну даними з основною логістичною програмною системою	Можливо
ФВ12	Забезпечення формування дворівневої позначки штрихового коду з трьома градаціями кольору	Можливо

Нефункціональні вимоги до програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору визначимо, як показано у табл. 4.2.

Таблиця 4.2. Базові нефункціональні вимоги, що пов’язані із застосуванням технології штрихового кодування

Код	Вимога	Пріоритет
НФВ1	Забезпечення можливості отримання даних, поданих у вигляді штрихового коду з трьома градаціями кольору, використовуючи обладнання різних типів (ШК-сканер, стаціонарний сканер, камера мобільного пристрою)	Бажано
НФВ2	Забезпечення можливості візуалізації даних, поданих у вигляді штрихового коду з трьома градаціями кольору, з використанням обладнання різних типів (ШК-принтер, стаціонарний принтер, екран мобільного пристрою)	Бажано
НФВ3	Забезпечення можливості вибору базового кольору для формування позначки на основі штрихових кодів з трьома градаціями кольору у бажаній гамі кольорів	Можливо
НФВ4	Автоматичне визначення напівтонового відтінку базового кольору для забезпечення формування контрастного зображення позначки на основі штрихових кодів з трьома градаціями кольору	Обов’язково (за умови виконання вимоги НФВ 3)
НФВ5	Забезпечення можливості аналізу основних кольорів дизайну носія для надання рекомендацій щодо вибору базового кольору та його напівтону для формування позначки на основі штрихових кодів з трьома градаціями кольору	Можливо

Передбачається, що зі системою можуть працювати три категорії користувачів, яких умовно назвемо «Оператор / Працівник складу», «Експедитор» та «Менеджер» (рис. 4.2 – рис. 4.4).

Оператор вводить первинні текстові дані про об’єкт обліку за допомогою відповідного інтерфейсу користувача.

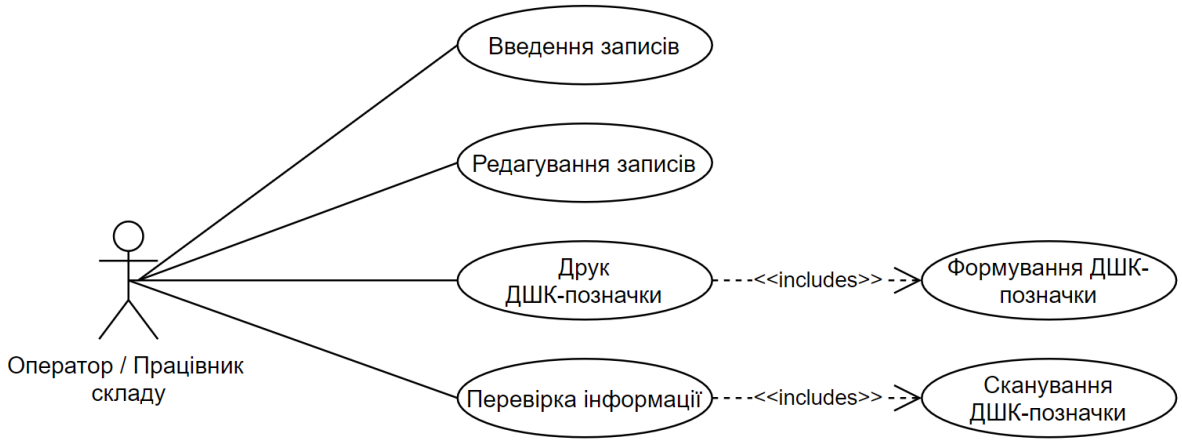


Рис. 4.2. Діаграма прецедентів для ролі «Оператор / Працівник складу»

Експедитор отримує доступ до даних про об’єкт обліку, використовуючи портативний пристрій, який дозволяє отримати доступ до «автономної бази даних», що міститься на нижньому рівні штрихкодової позначки. Це дозволяє отримувати доступ до загальної інформації про одиницю обліку в дорозі та у невеликих населених пунктах, де немає доступу до мережі Інтернет.

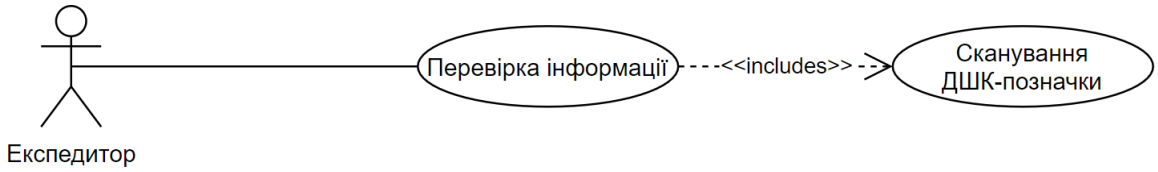


Рис. 4.3. Діаграма прецедентів для ролі «Експедитор»

Менеджер отримує доступ до даних про об’єкт обліку в умовах офісу, тобто за наявності доступу до мережі Інтернет. Тому він має можливість працювати з різнорідними файлами, які містять докладну та різнопланову інформацію про об’єкт обліку, що значно розширює можливості з виконання логістичних операцій. Доступ до докладної інформації про об’єкт обліку виконується шляхом сканування посилання, поданого у вигляді верхнього рівня штрихкодової позначки.

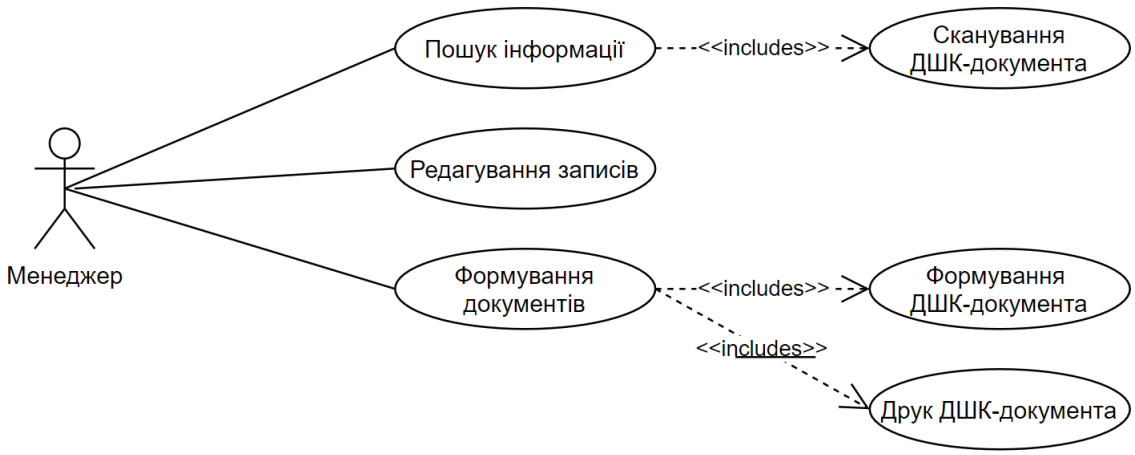


Рис. 4.4. Діаграма прецедентів для ролі «Менеджер»

Розробимо архітектуру програмної системи для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору, яка задовольняє вимогам, визначеним у табл. 4.1 і табл. 4.2, та забезпечує варіанти застосування, визначені відповідно до основних ролей користувачів цієї програмної системи.

4.2. Архітектура та компоненти програмної системи для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору

Враховуючи вищезазначені вимоги до програмної системи та особливості застосування технології штрихового кодування на основі штрихових кодів з трьома градаціями кольору, можна визначити базову архітектуру програмної системи, як показано на рис. 4.5.

Запропонована архітектура передбачає наступну логіку функціонування системи та її практичного використання.

На першому етапі роботи зі системою оператор вводить текстовий опис об’єкту обліку, що зберігатиметься в автономній базі даних у вигляді ДШК-позначки, якою промаркований сам об’єкт обліку, а також буде ключовим записом про цей об’єкт у хмарному сховищі даних. Хмарне сховище даних може бути реалізовано як сховище файлів. Після формування запису про об’єкт обліку може бути сформований логістичний документ за певним визначеним шаблоном. Цей документ пропонується представляти у форматі PDF, оскільки це дозволить здійснювати над ним різноманітні стандартні операції (друкування, надсилання електронною поштою тощо).

Після того як вхідні дані були введені до системи, відбувається підготовка до їх представлення у машиночитаному вигляді, а саме – у вигляді ДШК-позначки. Для цього спочатку відбувається аналіз вхідних даних, а далі – їх завадостійке кодування (з метою забезпечення цілісності даних та надійності їх подання) та формування ДШК-позначки. Позначку також пропонується зберігати у вигляді файлу формату PDF.

На етапі використання штрихового коду у вигляді позначки, нанесеної на об’єкт обліку, ДШК-позначка сканується, після чого залежно від режиму роботи (використання експедитором або менеджером)

здійснюється декодування даних верхнього або нижнього рівня та, відповідно до цього, відбувається або візуалізація на мобільному пристрої вмісту автономної бази даних, або доступ до документації про об’єкт, яка міститься у хмарному сховищі.

Ця логіка роботи відображена в архітектурі цієї системи. Як видно з рис. 4.5, схема включає в себе три основні модулі: «Опрацювання інформації про об’єкт логістики», «Штрихове кодування даних» та «Відновлення даних».

Модуль «Опрацювання інформації про об’єкт логістики» включає до себе компоненти, які забезпечують формування запису про об’єкт логістики, формування супровідної документації про об’єкт та цільове оброблення даних про об’єкт логістики. Ці компоненти забезпечують виконання вимог ФВ1, ФВ2, ФВ3, НФВ2, визначених у п. 4.1.

Модуль «Штрихове кодування даних» відповідає за аналізування вхідних даних, їх перетворення, яке включає ущільнення та завадостійке кодування, та подання у вигляді ДШК- або ДДШК-позначки відповідно до колірної гами, яка визначається на основі базового кольору, заданого при налаштуванні системи. Ці компоненти забезпечують виконання вимог ФВ4, ФВ5, ФВ6, ФВ7, ФВ12, НФВ2, НФВ3, НФВ4, НФВ5.

Модуль «Відновлення даних» дозволяє здійснити декодування штрихкодованих даних, отриманих з пристрою сканування (стаціонарний сканер, мобільний ШК-сканер, камера мобільного пристрою тощо), та отримати відновлені автономні дані, які містяться безпосередньо у позначці, або ж доступитися до сховища даних для отримання докладної інформації про об’єкт логістики. Ці компоненти забезпечують виконання вимог ФВ8, ФВ9, ФВ10, НФВ1.

До складу запропонованої архітектури також входять: сховище даних про об’єкти логістики, сховище службових даних (службовими даними є таблиці з вхідними алфавітами, параметри завадостійкого

кодування, налаштування базового кольору, інші налаштування системи), компонент для забезпечення цільового пошуку даних про об’єкт логістики.

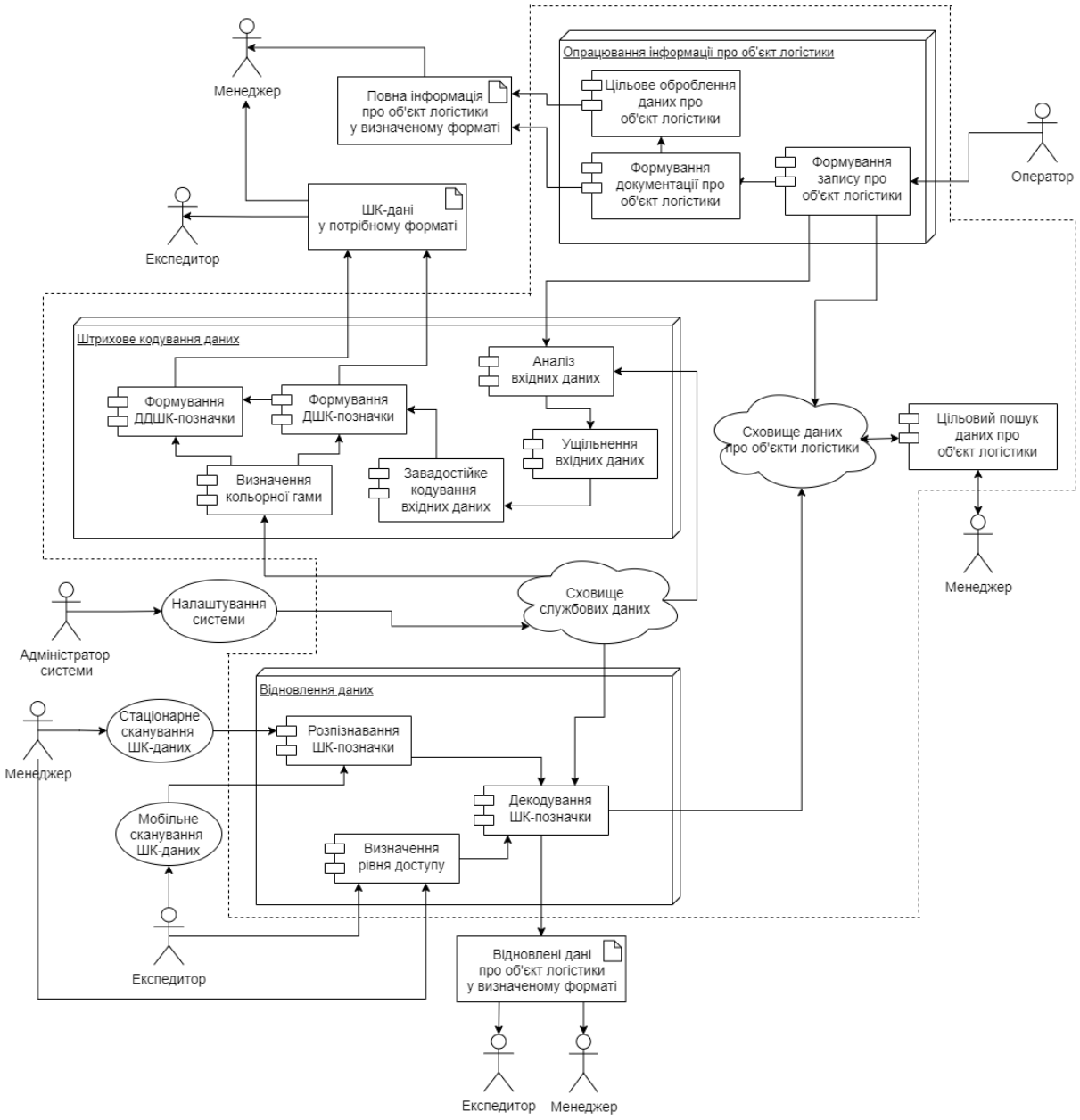


Рис. 4.5. Базова архітектура програмної системи для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору

Якщо програмна система для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору є окремою системою, що не

передбачає безпосередньої інтеграції з іншими логістичними програмними системами, то, окрім зазначених модулів, до архітектури цієї системи потрібно включити модуль цільової обробки інформації про логістичні об’єкти.

Запропонована архітектура програмної системи для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору є базовою, що уможливлює адаптування цієї архітектури до потреб логістичних компаній та дозволяє спростити процес розроблення програмного забезпечення для сфери логістики.

4.3. Шаблон проєктування програмного забезпечення для автоматичної ідентифікації об’єктів

Для спрощення процесу розроблення програмних систем, зокрема програмної системи для автоматичної ідентифікації на основі штрихових кодів з трьома градаціями кольору, доцільно використовувати шаблони проєктування.

Шаблоном проєктування [16, 61, 88, 104, 151] називають типовий спосіб вирішення певної проблеми, яка може часто зустрічатись при проєктуванні архітектури програмного забезпечення. Таким чином, процес розроблення програми може бути суттєво спрощений та пришвидшений завдяки використанню визначеної структури, яка доповнюється іншими програмними компонентами відповідно до предметної галузі.

Шаблони проєктування поділяються на три групи, відповідно до їхнього призначення: твірні шаблони, структурні шаблони та поведінкові шаблони.

Твірні шаблони забезпечують незалежність програмної системи від способу створення, композиції та представлення її об’єктів, що дозволяє гнучкіше працювати із можливими змінами до програмного продукту та

безпечно створювати нові об’єкти або навіть сімейства об’єктів, не руйнуючи вже побудовану архітектуру програмної системи.

Структурні шаблони вирішують задачу створення ієрархій класів, які будуть зручними для подальшої підтримки та дозволятимуть масштабувати програмну систему шляхом безпечного створення більших за розміром структур.

Поведінкові шаблони дозволяють ефективно розподілити обов’язки між об’єктами програмної системи для забезпечення ефективної та безпечної взаємодії між ними. На відміну від структурних шаблонів вони не тільки описують структуру та ієрархію об’єктів, але й визначають способи взаємодії різних об’єктів та класів.

На сьогодні відомо 23 класичних шаблони проєктування [151], що були розроблені групою авторів Gang of Four для вирішення типових проблем об’єктно-орієнтованого програмування. Разом із тим, існують десятки інших шаблонів проєктування [16, 61, 88, 104], запропонованих у відповідь на ті чи інші розповсюджені проблеми, які виникають у процесі розроблення програмного забезпечення в різних сферах людської діяльності. Тому наразі «шаблонний» підхід є вельми популярним не тільки серед фахівців об’єктно-орієнтованого програмування, але й в усіх інших галузях програмування.

Зокрема, на окрему увагу заслуговує клас шаблонів програмного забезпечення, які дістали назву архітектурних [16, 88, 104]. На відміну від описаних раніше класичних шаблонів проєктування, архітектурні шаблони вирішують проблеми вищого рівня – рівня архітектури системи, маючи у фокусі уваги не просто об’єкти та класи, а окремі частини певної програмної системи.

У сфері логістики існують різнопланові проблеми, які потребують інноваційних програмних рішень. Зокрема, дослідження в рамках цієї

дисертаційної роботи сфокусоване на проблемі автоматичної ідентифікації об’єктів логістики.

Якщо розглядати це питання з точки зору розроблення програмного забезпечення автоматичної ідентифікації об’єктів, постає доволі принципова проблема, пов’язана із прийманням та передачею даних у закодованому та декодованому вигляді між джерелом цих даних та їхнім майбутнім приймачем. Розглянемо це на прикладі.

Нехай існує система ідентифікації логістичних об’єктів, промаркованих кодом EAN, якою користується невелика логістична компанія. На поточний момент компанії цього достатньо для обліку об’єктів та їх ефективного транспортування. Менеджери використовують стаціонарні принтери EAN-кодів та стаціонарні сканери, тоді як експедитори застосовують портативні сканери. Програмна система, яку використовує ця логістична компанія, працює з даними відповідного формату та з EAN пристроями.

Згодом логістична компанія почала розширятись й запрошувати на роботу більше експедиторів та менеджерів, що спричинило необхідність закуповувати додаткове обладнання. До того ж, керівництво компанії вирішило, що для певних об’єктів ефективніше буде використовувати QR-коди замість EAN-кодів, адже вони дозволяють кодувати не тільки цифри, але й текстову інформацію, що дозволить закласти важливу інформацію у сам код без доступу до бази даних.

Для реалізації цієї стратегії необхідно внести зміни у програмну систему, щоб, по-перше, додати модуль кодування та декодування даних у форматі, прийнятному для QR-кодів, по-друге, уможливити отримання даних з іншого джерела – камери смартфону та передача декодованих даних на інший приймач – смартфон або комп’ютер.

Вносити зміни у суцільну програмну систему є вкрай нераціональним рішенням, оскільки фінансові витрати на

перепроєктування програмного забезпечення потенційно можуть перебільшити будь-які переваги переходу з EAN-кодів на QR-коди. До того ж, в перспективі може виникнути потреба перейти до використання інших типів штрихових кодів, наприклад, ДДШК з трьома градаціями кольору, який пропонується у цій дисертаційній роботі, або ж до принципово іншої технології ідентифікації логістичних об’єктів – радіочастотної ідентифікації, що потребуватиме підключення додаткового апаратного забезпечення, а також розроблення специфічних алгоритмів кодування та декодування. Відповідно, система має бути якомога гнучкішою, щоб розробник мав можливість легко додавати до неї нові компоненти, не витрачаючи час на масштабний рефакторинг цієї програмної системи.

Найкращим рішенням цієї проблеми є використання на етапі проєктування логістичної системи такого шаблону проєктування, який дозволив би гнучко працювати з її структурою, концептуальна ідея якої показана на рис. 4.6.

У цьому дисертаційному дослідженні пропонується шаблон проєктування програмного забезпечення «Перетворювач» або «Конвертер» (Converter Design Pattern), мета якого – ефективна та безпечна взаємодія між модулями отримання даних з джерела, обробки даних (тобто власне кодування/декодування) та передачі даних на приймач. Таку назву шаблон дістав, оскільки застосовувати його є сенс тоді, коли постає задача не просто отримання даних з джерела та їх передачі приймачу, але й конвертації цих даних відповідно до потреб предметної галузі, в результаті чого дані, отримані на вході, перетворюються у якісно іншу форму, яка передається приймачу для подальшого використання у визначеній формі оператором цієї програмної системи.

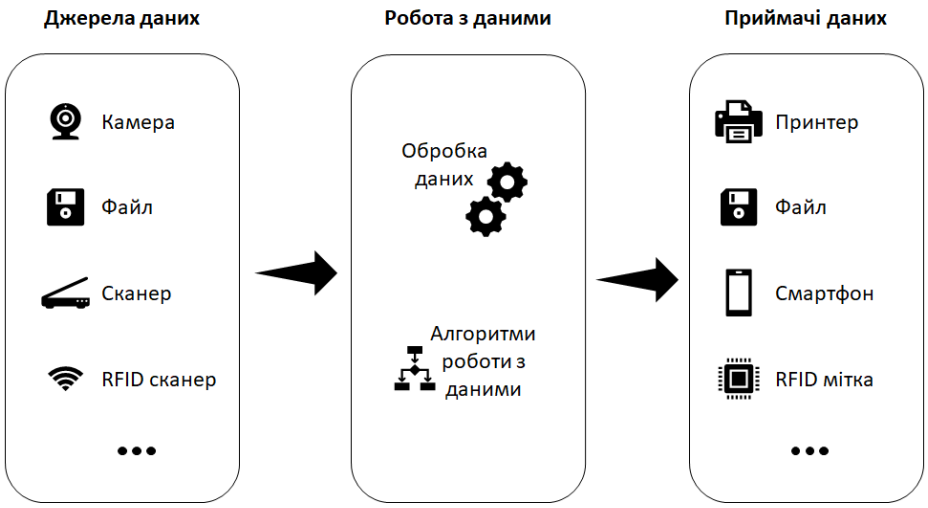


Рис. 4.6. Концептуальна схема програмної системи

Шаблон «Перетворювач» пропонує виділити джерело та приймач в окремі класи, які взаємодіятимуть з основним класом-обробником, що відповідатиме за оброблення даних, отриманих від джерела, з метою їх подальшої передачі приймачу.

Важливо, щоб всі джерела мали спільний інтерфейс, так само як й всі приймачі матимуть свій спільний інтерфейс. Використовуючи цей інтерфейс, програмна система зможе оперувати будь-якими пристроями надходження даних та отримання даних, а також комбінувати їх залежно від того, яке обладнання є зручнішим для кожного конкретного практичного випадку.

Водночас, поведінка обробника жодним чином не залежить ані від джерела, ані від приймача, що даватиме змогу легко додавати нові модулі оброблення даних незалежно від того, звідки ці дані надходять і куди мають бути надіслані після відповідних дій над ними.

На рис. 4.7 показана структура шаблону проєктування «Перетворювач» у загальному вигляді.

Розглянемо структурні складові цього шаблону проєктування.

1. **Клієнт** (*Client*) – це клас, що містить безпосередню бізнес-логіку програмної системи відповідно до предметної галузі.

2. **Обробник** (*Handler*) – це клас, який працює одночасно із клієнтом, джерелом даних та приймачем даних. У разі необхідності клієнт може створити декілька обробників одночасно, що може стати в нагоді, якщо дані (у тому числі різні дані) потрібно обробити декількома способами (наприклад, якщо розглядати логістичну систему – закодувати дані у вигляді QR-коду та RFID-мітки, якщо з певних причин об'єкт логістики маркується у декілька способів).

3. **Додаткові обробники** (*Additional Handlers*) містять різні варіації оброблення даних залежно від потреб програмної системи. У загальному випадку можуть бути відсутні.

4. **Джерело** (*Source*) описує загальний інтерфейс для всіх можливих джерел даних. Всі описані в ньому методи будуть доступні з класу обробника та його підкласів.

5. **Конкретні джерела** (*Concrete Sources*) реалізують отримання даних з джерел, використання яких передбачається у цій програмній системі. Робота з обробником через загальний інтерфейс дозволяє легко додавати нові джерела у разі, якщо у системі з'явиться така потреба.

6. **Приймач** (*Receiver*) описує загальний інтерфейс для всіх можливих приймачів даних. Так само як й у випадку джерела, всі описані в ньому методи будуть доступні з класу обробника та його підкласів.

7. **Конкретні приймачі** (*Concrete Receivers*) реалізують передавання даних на будь-який зареєстрований у системі приймач. Завдяки роботі обробника через загальний інтерфейс можна безпечно додавати нові приймачі даних у разі, якщо у системі з'явиться така необхідність.

8. **Синхронізатор** (*Synchronizer*) дозволяє у разі необхідності перетворювати («синхронізовувати») дані у форматі джерела на тип даних, з яким працює той чи інший обробник, або навпаки – формат обробника перетворити на тип даних, який сприймає приймач. Він описує загальний

інтерфейс всіх можливих синхронізаторів. Всі описані в ньому методи будуть доступні з підкласів обробника.

9. **Конкретні синхронізатори** (*Concrete Synchronizers*) реалізують процедуру перетворення даних з одного типу на інший тип. Завдяки тому, що обробники працюють з синхронізаторами через загальний інтерфейс, можна у разі необхідності додавати нові синхронізатори для нових джерел/приймачів або нового обробника, а також створювати комбінації перетворень залежно від типів, з якими працюють джерела, приймачі й обробники.

У лістингах 4.1–4.5 мовою C# наведено фрагменти програмного коду, які ілюструють кожну зі складових шаблону проєктування «Перетворювач». Повний код шаблону в узагальненому вигляді наведено у Додатку Ж.

У лістингу 4.1 наведено фрагмент ієрархії джерел даних. Інтерфейс містить метод зчитування з джерела (камери, сканеру, файлу тощо), який реалізується у класах конкретних джерел. Варто зауважити, що залежно від джерела отримані з нього дані можуть бути в різних формах.

Класи конкретних джерел за необхідністю можуть бути розширені додатковими методами, необхідними для успішного отримання інформації з джерела.

Лістинг 4.2 демонструє фрагмент ієрархії класів приймачів даних. Інтерфейс приймачів складається з методу передачі (записування) даних на приймач. Залежно від конкретного приймача результатом методу може бути як файл певного формату, так й подія, наприклад, друк на принтері. У разі потреби класи конкретних приймачів можуть бути розширені додатковими методами, необхідними для передачі даних на приймач.

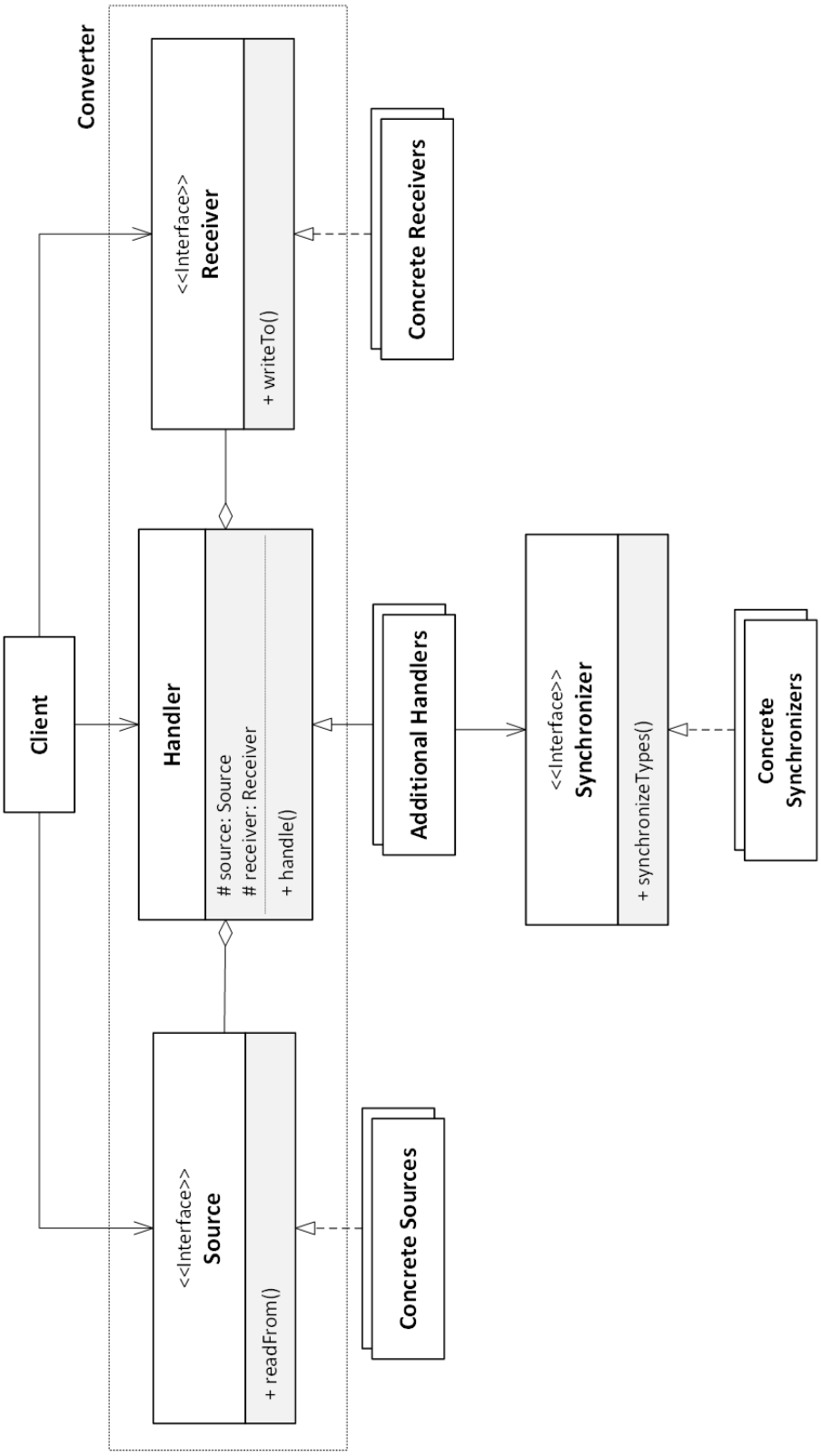


Рис. 4.7. Структура шаблону проєктування «Перетворювач»

Лістинг 4.1. Фрагмент ієрархії класів джерел даних

```
/// <summary>
/// The Source interface defines the must-have operations for any sources
/// connected to the program system.
/// </summary>
interface ISource
{
    void ReadFrom();
}
/// <summary>
/// Concrete Sources implement reading depending on a specific source
/// that is being connected to the system.
/// </summary>
class ConcreteSource : ISource
{
    public void ReadFrom()
    {
        throw new NotImplementedException();
    }
}
```

Лістинг 4.2. Фрагмент ієрархії класів приймачів даних

```
/// <summary>
/// The Receiver interface defines the must-have operations for any receivers
/// connected to the program system.
/// </summary>
interface IReceiver
{
    void WriteTo();
}

/// <summary>
/// Concrete Receivers implement writing data to a receiver depending
/// on the specific receiver that is being connected to the system.
/// </summary>
class ConcreteReceiver : IReceiver
{
    public void WriteTo()
    {
        throw new NotImplementedException();
    }
}
```

У лістингу 4.3 наведено фрагмент ієрархії класів синхронізаторів даних. Інтерфейс містить метод реалізації перетворення даних з одного типу на інший. Якщо необхідності у синхронізації типів даних немає, синхронізатори можуть практично не застосовуватись. Водночас, у разі необхідності кожний синхронізатор може бути розширений додатковими методами для представлення даних одного типу у вигляді даних іншого типу.

Лістинг 4.3. Фрагмент ієрархії класів синхронізаторів даних

```
/// <summary>
/// If applicable, The Synchronizer defines how to convert the source data type
/// into the data type the Handler can process.
/// </summary>

interface ISynchronizer
{
    void SynchronizeTypes();
}

/// <summary>
/// Concrete Synchronizers implement data synchronization procedure for each
/// specific pair of data types.
/// </summary>
class ConcreteSynchronizer : ISynchronizer
{
    public void SynchronizeTypes()
    {
        throw new NotImplementedException();
    }
}
```

Лістинг 4.4 наводить фрагмент ієрархії класів обробників даних. Особливістю цієї ієрархії є те, що базовий клас здебільшого не містить алгоритмів оброблення даних, які реалізуються у підкласах, кожний з яких відповідає за певний модуль оброблення даних. Разом із тим, методи отримання даних та їх передачі на приймач є спільними для всіх обробників. Як окремий випадок, в деяких системах додаткових обробників може й не бути, або навпаки, може існувати спільний етап оброблення даних, який, у такому разі, буде винесено у базовий клас обробників. У наведеному лістингу, окрім головного методу `Handle()`, показані також допоміжні методи додаткових обробників: синхронізація типів та перевірка, чи вона взагалі потрібна. Залежно від складності головного методу оброблення, а також необхідності під’єднання сторонніх бібліотек, клас додаткового обробника може бути розширений необхідними для здійснення процедури оброблення даних методами.

Лістинг 4.4. Фрагмент ієрархії класів обробників даних

```
/// <summary>
/// The Handler defines a base class that extracts raw data from a source
/// and passes processed data to a receiver.
/// </summary>
class Handler
{
    public void GetDataFromSource(ISource source)
    {
        throw new NotImplementedException();
    }

    public void OutputConvertedData(IReceiver receiver)
    {
        throw new NotImplementedException();
    }
}

class AdditionalHandler : Handler
{
    public AdditionalHandler(ISource source)
    {
        GetDataFromSource(source);
        if (IsSynchronizationRequired())
        {
            SynchronizeTypes();
        }
        HandleData();
    }

    private void Handle()
    {
        throw new NotImplementedException();
    }

    private bool IsSynchronizationRequired()
    {
        throw new NotImplementedException();
    }

    private void SynchronizeTypes()
    {
        ISynchronizer synchronizer = new ConcreteSynchronizer();
        synchronizer.SynchronizeTypes();
    }
}
```

У лістингу 4.5 наведено фрагмент коду клієнту програмної системи. Клієнт працює з джерелом та приймачем, яким у реальній системі можуть бути передані параметри та налаштування для коректної роботи зчитування та передавання даних. Крім того, клієнт працює з обробником даних. У разі необхідності може здійснюватися робота з даними, отриманими з різних джерел, які потребують різних методів оброблення.

Лістинг 4.5. Фрагмент коду клієнту програмної системи

```
/// <summary>
/// The Client class defines the domain logic of the program system.
/// </summary>
class Client
{
    ISource source;
    IReceiver receiver;

    public void ClientLogic()
    {
        source = new ConcreteSource();
        receiver = new ConcreteReceiver();

        Handler handler = new AdditionalHandler(source);
        handler.OutputConvertedData(receiver);
    }
}
```

Повернемось до практичного прикладу логістичної програмної системи. В цій системі можна формувати та розшифровувати штрихкодові позначки двох типів – ДДШК та QR-коди. Крім того, система вже працює з камерою та файлами як джерелом отримання даних, а також з принтером та смартфоном як потенційними приймачами закодованих/декодованих даних. Відповідно, мають бути реалізовані синхронізатори форматів даних для таких джерел та приймачів. Втім, передбачається й можливість додавання нових пристроїв-джерел, пристроїв-приймачів, а також розширення функціональності шляхом додавання нових різновидів кодів з відповідними алгоритмами кодування/декодування.

На рис. 4.8 у вигляді спрощеної UML-діаграми представлено адаптований до конкретної програмної системи шаблон проєктування «Перетворювач».

У разі розширення системи на використання відсутніх наразі пристроїв (як джерела або як приймача) розробник зможе легко додати відповідні класи, не порушуючи логіки вже існуючих програмних компонентів цієї системи.

Таким чином, можна сформулювати наступні можливі випадки, коли застосування шаблону «Перетворювач» буде доцільним.

– Програмний код містить декілька різних реалізацій функціональності одного типу (як-от реалізація оброблювача даних відповідно до типу штрихового коду), які можуть використовувати різні реалізації іншого типу (наприклад, різні джерела та різні приймачі), не змінюючи при цьому своєї внутрішньої поведінки. Перетворювач дозволяє рознести ці реалізації в окремі ієрархії та комбінувати їх відповідно до логіки програмної системи.

– Необхідно працювати з одними й тими ж даними, які можуть бути представлені в різних форматах (наприклад, ДДШК може бути представлений у вигляді графічної позначки, що зчитується камерою, але при цьому для роботи із ним необхідно перетворити графічну позначку на матричний формат, з яким працює оброблювач кодів такого типу). Перетворювач дозволяє вирішити проблему несумісності типів завдяки наявності окремої ієрархії синхронізаторів, використовуючи яку можна працювати з різними джерелами та різними оброблювачами незалежно від типу даних, з яким кожний з них оперує.

Таким чином, шаблон проєктування «Перетворювач» дозволяє:

- комбінувати різні джерела, обробники та приймачі даних, а отже – використовувати один й той самий код для створення різних конфігурацій певної програмної системи;
- ізолювати програмний код отримання даних або їх передачі певному приймачу від основної логіки оброблення цих даних;
- відокремити та приховати від оброблювача даних подробиці перетворення типів даних.

Водночас, подібно більшості шаблонів проєктування шаблон «Перетворювач» суттєво ускладнює структуру програмного коду внаслідок введення додаткових класів.

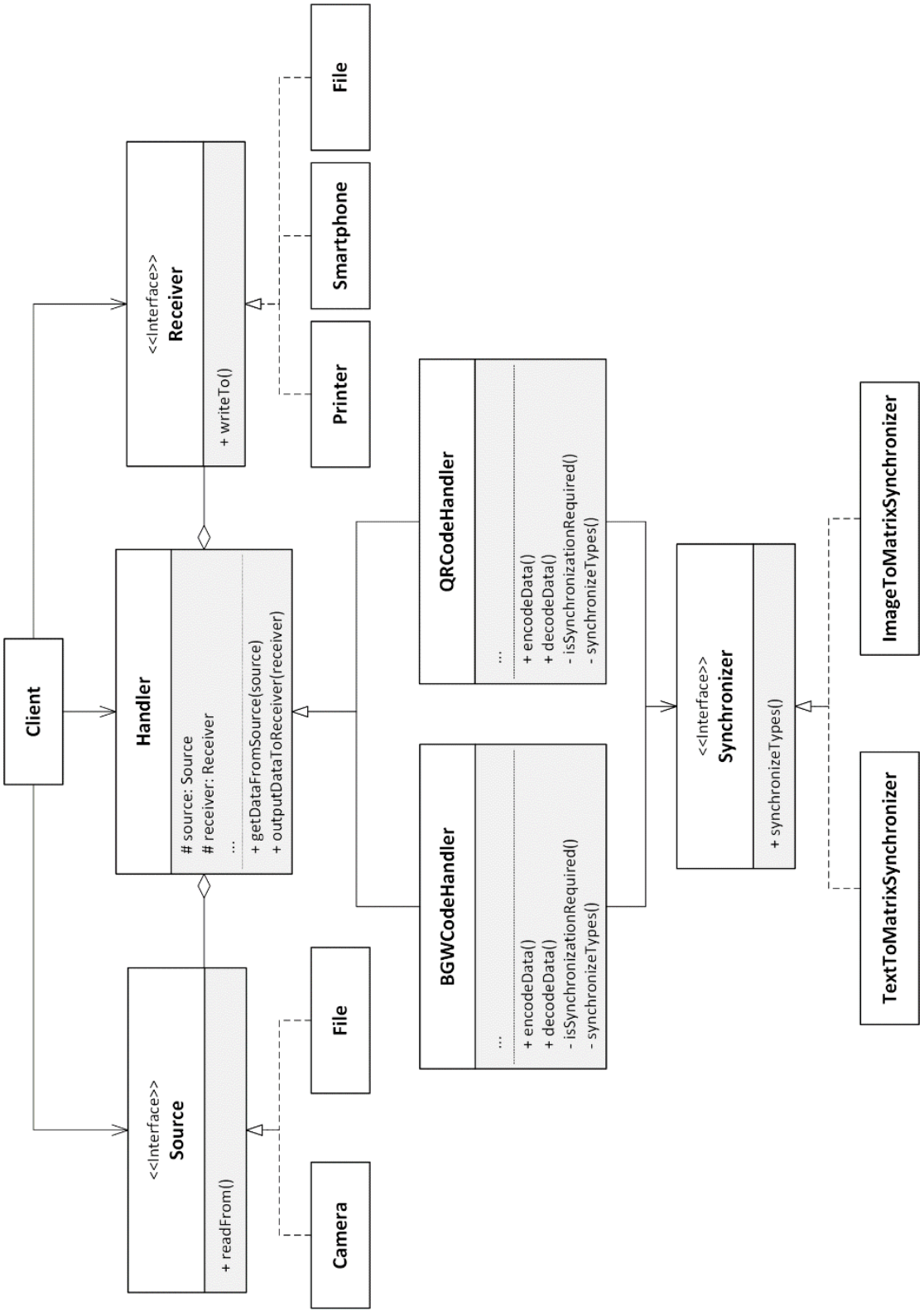


Рис. 4.8. Приклад застосування шаблону «Перетворювач» для логістичної програмної системи

Варто також зазначити, що Перетворювач може бути реалізований шляхом комбінування декількох класичних шаблонів проєктування GoF.

Так, проблема несумісності даних може бути реалізована через шаблон проєктування «Адаптер».

Використання шаблону проєктування «Міст» дасть можливість відокремити абстракцію (тип пристою) від реалізації (як з цього пристрою зчитуються або на нього записуються дані). Альтернативою може також бути шаблон проєктування «Стратегія».

За наявності у розроблюваній програмній системі спільної для будь-якого типу штрихового коду функціональності додаткові обробники можуть бути реалізовані із застосуванням шаблону проєктування «Декоратор» як обгортки основного обробника.

Можливе застосування шаблону проєктування «Шаблонний метод», який дозволяє підкласам базового класу змінювати реалізацію кроків базового алгоритму, не змінюючи при цьому структуру базового класу. Однак, цей шаблон передбачає наявність спільних кроків алгоритмів базового класу та підкласів, тоді як у шаблоні проєктування «Перетворювач» може й не бути спільного алгоритму перетворення даних.

Таким чином, шаблон проєктування програмного забезпечення «Перетворювач» пропонує таку організацію структури програмного коду, яка дозволяє суттєво спростити процес розроблення та подальшого розширення системи логістики або будь-якої іншої програмної системи, що має подібну логіку.

4.4. Технологія проєктування програмних засобів для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору

Програмні засоби для автоматичної ідентифікації об’єктів логістики

на основі штрихових кодів з трьома градаціями кольору можуть бути окремими програмними системами або входити до складу інших систем, наприклад, ERP-систем.

Розглянемо основні (укрупнені) етапи проєктування окремої програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору.

На першому етапі проєктування такої системи потрібно визначити місце застосування елементів технології штрихового кодування у логічному ланцюгу передбачених процедур оброблення логістичної інформації. Виходячи з цього, необхідно сформулювати вимоги до програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору з урахуванням шкали пріоритетів, запропонованої у п. 4.1.

Визначені вимоги необхідно врахувати при адаптації запропонованої базової архітектури програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору до потреб кінцевих користувачів розроблюваної програмної системи.

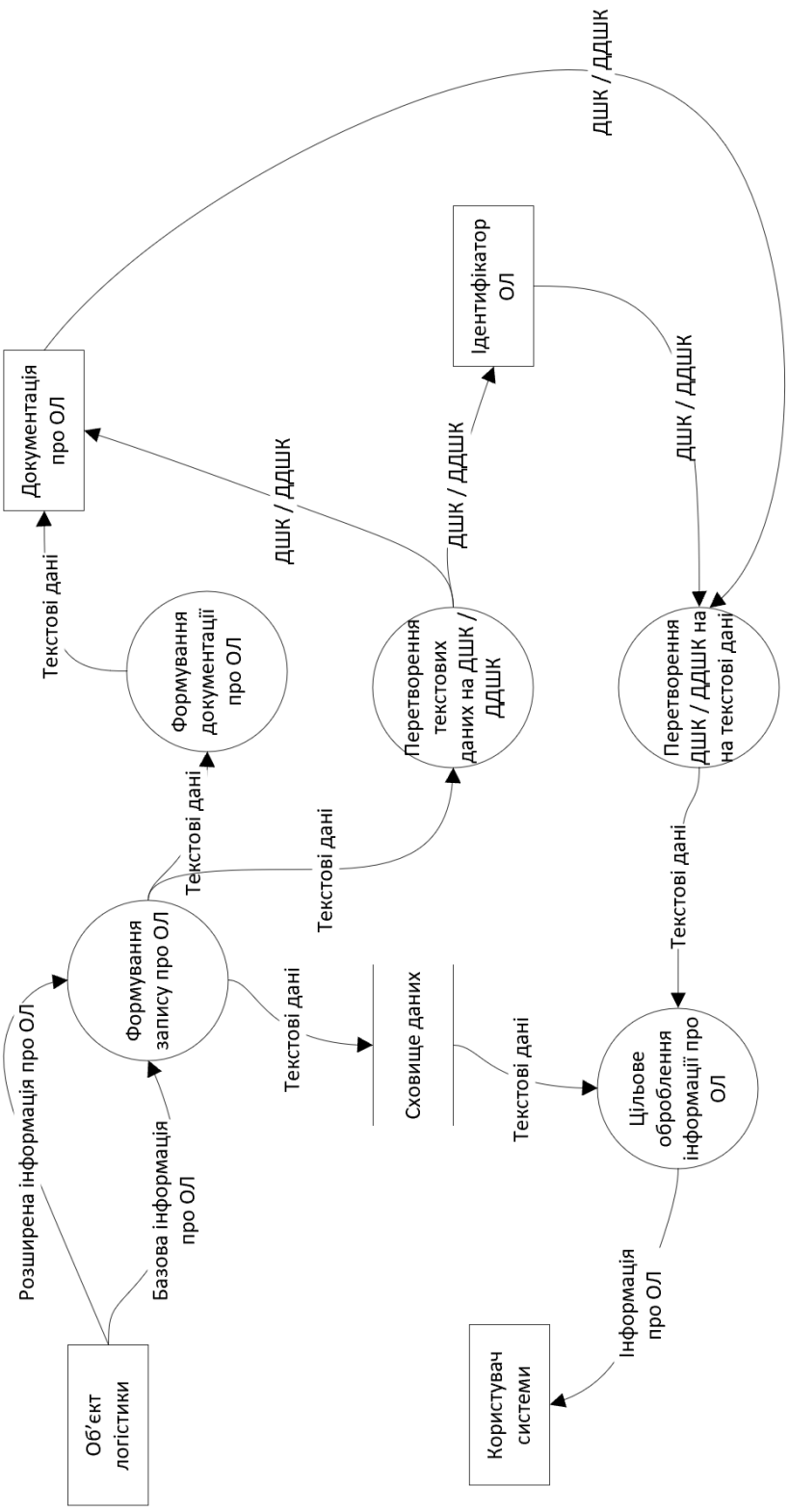


Рис. 4.9. Типова діаграма потоків даних

Також потрібно визначити потоки даних у розроблюваній системі. На рис. 4.9 наведено типову діаграму потоків даних для базової архітектури програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору із зазначенням інформації, що отримується зі зовнішніх джерел або надається користувачам системи у тому чи іншому вигляді, та типу даних (ДШК, ДДШК, текстові), що оброблюються у системі.

Виходячи з вимог щодо формату подання інформації про об’єкт логістики, необхідно визначити формат документації про об’єкт логістики. Приклад логістичного документу, який може бути використаний експедитором при транспортуванні об’єкту, наведено у Додатку З.

Також на першому етапі потрібно проаналізувати умови фізичного використання штрихкодових позначок для визначення вимог до забезпечення компактності та завадостійкості подання інформації про об’єкт логістики.

На другому етапі проєктування програмної системи необхідно визначити оптимальні вхідні алфавіти, які мають мінімально можливий розмір, проте дозволятимуть подання всієї необхідної інформації про логістичний об’єкт, виходячи з вимог до формату подання даних при формуванні логістичної документації та ідентифікаторів (позначок) логістичних об’єктів всіх типів, що передбачається застосовувати у програмній системі.

Ґрунтуючись на визначених вхідних алфавітах, потрібно розробити алгоритмічне забезпечення для ущільненого подання даних у вигляді штрихового коду з трьома градаціями кольору, а також алгоритмічне забезпечення для забезпечення завадостійкості даних, поданих у вигляді ШК-позначки.

На третьому етапі необхідно виконати проєктування компонент програмної системи, а саме:

- спеціальних компонент, які реалізовуватимуть алгоритмічне забезпечення для ущільненого подання даних у вигляді штрихового коду з трьома градаціями кольору та алгоритмічне забезпечення для забезпечення завадостійкості даних, поданих у вигляді ШК-позначки;
- універсальних компонент, які реалізовуватимуть основні функції системи з цільового оброблення даних про об’єкт логістики, формування логістичної документації, роботи зі сховищем даних тощо.

У разі проєктування програмних засобів для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, які є складовою частиною інших систем, у складі архітектури такого програмного забезпечення потрібно передбачити компоненти, що забезпечуватимуть інтеграцію розроблюваного програмного засобу з основною логістичною програмною системою.

Процес розроблення програмної системи для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору доцільно виконувати з використанням шаблону проєктування «Перетворювач», запропонованого у цій дисертаційній роботі.

4.5. Висновки до четвертого розділу

Виконане дослідження, результати якого представлені у цьому розділі, дозволяє зробити наступні висновки.

1. Запропоновано базову архітектуру програмної системи, яка ґрунтується на використанні технології штрихового кодування, а саме – на концепції дворівневого штрихового коду, яка пропонується у цьому дослідженні. Ця базова архітектура включає універсальний набір

модулів, що забезпечують реалізацію основних операцій зі сфері логістики.

2. Для розроблення конкретної логістичної програмної системи достатньо адаптувати запропоновану базову архітектуру до потреб конкретної логістичної компанії. Такий підхід дозволяє спростити процес розроблення програмного забезпечення для галузі логістики.
3. Запропоновано шкалу пріоритизації базових вимог до програмної системи, зокрема програмної системи для автоматичної ідентифікації об’єктів логістики, що дозволяє формулювати вимоги двох категорій, а саме – залежні та незалежні вимоги, з врахуванням взаємозв’язку між ними, що сприятиме оптимізації процесу проєктування програмних систем.
4. Розроблено шаблон проєктування, який дозволяє комбінувати різні реалізації різних типів, ізолювати код отримання даних або передачі даних певному приймачу від основної логіки оброблення цих даних, відокремити та приховати від обробника даних подробиці перетворення типів даних, що зі свого боку уможливлює спрощення процесу розроблення та подальшого розширення програмної системи.
5. Запропоновано основні (укрупнені) етапи технології проєктування програмних систем для автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальну науково-технічну задачу підвищення ефективності розроблення алгоритмічного та програмного забезпечення процесів автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору шляхом вдосконалення технології проєктування програмних систем та розроблення нового алгоритмічного і програмного забезпечення технології штрихового кодування даних, що забезпечує спрощення процесу розроблення програмного забезпечення у галузі логістики.

У результаті дисертаційного дослідження отримано такі основні наукові результати:

1. Показано, що доцільно подавати дані про об’єкти логістики у вигляді штрихових кодів з трьома градаціями кольору, що потребує розроблення нового алгоритмічного та програмного забезпечення ущільнення та завадостійкості даних, яке ґрунтуватиметься на використанні трійкової системи числення та виконанні обчислень у скінченому полі $GF(3^s)$.
2. Розроблено базову архітектуру програмної системи автоматичної ідентифікації об’єктів на основі штрихових кодів з трьома градаціями кольору, використання якої дозволяє спростити процес розроблення програмного забезпечення систем автоматичної ідентифікації об'єктів. Показано, що запропонована базова архітектура уможливлює забезпечення двох рівнів доступу до інформації про об’єкт логістики та підвищення щільності подання даних завдяки використанню дворівневих штрихових кодів з трьома градаціями кольору.
3. Розроблено шаблон проєктування програмного забезпечення «Перетворювач», характерною рисою якого є можливість ізолювання програмного коду, який забезпечує отримання даних або передачі

даних приймачу, та програмного коду, який забезпечує перетворення типів даних, від основної логіки оброблення даних. Продемонстровано, що застосування розробленого спеціалізованого шаблону проєктування уможливлює спрощення процесу розроблення та подальшого розширення програмної системи автоматичної ідентифікації об’єктів.

4. Запропоновано шкалу пріоритизації базових вимог до програмної системи, зокрема програмної системи для автоматичної ідентифікації об’єктів логістики, що дозволяє формулювати вимоги двох категорій: залежні та незалежні вимоги, з врахуванням взаємозв’язку між ними, що сприятиме оптимізації процесу проєктування програмних систем.
5. Запропоновано основні (укрупнені) етапи технології проєктування програмних систем автоматичної ідентифікації об’єктів логістики на основі штрихових кодів з трьома градаціями кольору, яка дозволяє спростити процес створення спеціалізованих логістичних програмних систем.
6. Показано, що формування двовимірної штрихкодової позначки у трьох градаціях кольору дозволяє компактне подання інформації про об’єкт логістики у машиночитаному вигляді. Розроблено алгоритмічне забезпечення та програмне забезпечення процесів формування даних у вигляді двовимірних штрихових кодів, яке, на відміну від існуючих, ґрунтується на використанні процедури виконання операцій над даними у трійковій системі числення та знаходженні парето-оптимальних розв’язків для визначення потужності алфавіту, що уможливлює формування символіки штрихового коду відповідно до вимог галузі застосування та підвищення щільності подання даних на носії у середньому в 1,16–1,5 разів.
7. Показано, що для забезпечення розмежування доступу до штрихкодованих даних у логістичній програмній системі доцільно формувати дані про об’єкт логістики у вигляді дворівневої двовимірної

штрихкодової позначки. Розроблено алгоритмічне та програмне забезпечення процесів формування дворівневих штрихкодових позначок з трьома градаціями кольору, яке, на відміну від існуючих, ґрунтується на використанні процедури визначення контрольного біту та процедури створення штрихкодової позначки у трьох градаціях кольору, що уможливлює два рівні доступу до інформації про об’єкт логістики.

8. Показано, що штрихкодовані дані можуть бути ушкоджені внаслідок спотворень двох типів – помилок та стирань. Розроблено алгоритмічне та програмне забезпечення завадостійкості позначок двовимірних штрихових кодів, яке, на відміну від існуючого, ґрунтується на застосуванні процедури виконання обчислень у скінченому полі $GF(3^s)$, що дозволяє забезпечити відновлення даних при ушкодженні до 37,5 % площі штрихкодової позначки у трьох градаціях кольору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Antonio Grillo, Alessandro Lentini, Marco Querini, Giuseppe F. Italiano. High Capacity Colored Two Dimensional Codes. Proceedings of the IEEE International Multiconference on Computer Science and Information Technology. 2010. P. 709–716.
2. Barcode types. https://www.activebarcode.com/codes/
3. Blelloch G. E. Introduction to Data Compression. Carnegie Mellon University, 2001. 54 p.
4. CargoWise. https://www.cargowise.com/
5. Castro J., Kolp M., Mylopoulos J. Towards requirements-driven information systems engineering: the Tropos project. Information Systems, 2002. Vol. 27(6), P. 365–389.
6. Chen J., Zhao W. Logistics automation management based on the Internet of things. Cluster Computing 2019, Vol. 22, P. 13627–13634. https://doi.org/10.1007/s10586-018-2041-2
7. Chen L., A directed graphical model for linear barcode scanning from blurred images. Asian Conf. on Computer Vision, Springer, 2012. P. 524–535.
8. Chung L., do Prado Leite J.C.S. On Non-Functional Requirements in Software Engineering. Lecture Notes in Computer Science. Springer, 2009. Vol. 5600. doi.org/10.1007/978-3-642-02463-4_19
9. Code 128/GS1-128 Barcode. https://www.barcodefaq.com/1d/code-128/#Code-128CharacterSet
10. Code metrics values. Microsoft, 2018. https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019
11. CodeTwo QR Code Desktop Reader & Generator. https://www.codetwo.com/freeware/qr-code-desktop-reader
12. Create QR Code. https://createqrcode.appspot.com

13. Cysneiros L.M., do Prado Leite J.C. Using UML to reflect non-functional requirements. Proceedings of the 2001 Conference of the Centre For Advanced Studies on Collaborative Research. IBM Centre for Advanced Studies Conference. IBM Press, 2001. Vol. 2.
14. De Santis R., Montanari R., Vignali G., Bottani E. An adapted ant colony optimization algorithm for the minimization of them travel distance of pickers in manual warehouses. Eur. Journal of Oper. Res., 2017. Vol. 267(1), P. 120–137.
15. De Sousa T. G. M. C., Castro J.F.B. Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle. Workshop em Engenharia de Requisitos, Piracicaba-SP, 2003, P. 223–239.
16. Dmitri Nesteruk. Design Patterns in .NET: Reusable Approaches in C# and F# for Object-Oriented Software Design. Apress. 2019. 356 p.
17. Dutoit A. H., Paech B. Rationale-based use case specification. Requirements engineering, 2002. Vol. 7(1), P. 1–3.
18. Dutoit A. H., McCall R., Mistrík I., Paech B. Rationale Management in Software Engineering. Springer, 2006.
19. EAN-13 Barcode Specifications. https://barcode1.com.au/ean-13-specifications
20. Fitzgerald B., Stol K. J. Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 2017. Vol. 123, P.176–189.
21. Fleck G., Moser M., Pichler J. Improving Quality of Data Exchange Files. An Industrial Case Study. Springer, Lecture Notes in Computer Science, 2019. Vol. 11915, P. 161–175. https://doi.org/10.1007/978-3-030-35333-9_12.
22. Gabriel Baptista, Francesco Abbruzzese. Hands-On Software Architecture with C# 8 and .NET Core 3: Architecting software solutions using microservices, DevOps, and design patterns for Azure Cloud. Packt Publishing. 2019. 598 p.

23. Ghanbari H., Vartiainen T., Siponen M. Omission of quality software development practices: a systematic literature review. ACM Comput. Surv., 2018. Vol. 51, Issue 2, P. 1–27.
24. Glinz M. On Non-Functional Requirements. Proceedings of the 15th IEEE International Requirements Engineering Conference, 2007. P. 21–26.
25. Gross D., Yu E. From Non-Functional Requirements to Design through Patterns. Requirements Eng. 2001. Vol. 6, P. 18–36. doi.org/10.1007/s007660170013.
26. GS1 General Specifications. Release 20.0, January 2020. 488 p. https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf
27. Gunasekaran A., Subramanian N., Papadopoulos T. Information technology for competitive advantage within logistics and supply chains: a review. Transp. Res., 2017. Vol. 99. P. 14–33.
28. Gunter C., Gunter E., Jackson M., Zave P. A Reference Model for Requirements and Specifcations. IEEE Software, 2000. P. 37–43.
29. Halstead M. H., McCabe T. A. Software Complexity Measure. IEEE Transactions on Software Engineering, 1976. Vol. 2, No. 12, P. 308–320.
30. Hekimian-Williams C., Grant B., Liu X., Zhang Z., Kumar P. Accurate localization of RFID tags using phase difference. Proceedings of the IEEE International Conference on RFID, 2010, P. 89–96.
31. Hekler E.B., Klasnja P., Riley W.T., Buman M.P., Huberty J., Rivera D.E., Martin C.A. Agile science: creating useful products for behavior change in the real world. Transl. Behav. Med. 2016. Vol. 6(2), P. 317–328.
32. Hua Yang et al. Automatic barcode recognition method based on adaptive edge detection and a mapping model. Journal of Electronic Imaging, 2016. Vol. 25(5), 16 p.
33. Iacob C., Harrison R. Retrieving and analyzing mobile apps feature requests from online reviews. Proceeding of the 10th IEEE Working Conference on Mining Software Repositories, 2013. P. 41–44.

34. Ibraheem N. A., Hasan M. M., Khan R. Z., Mishra P. K. Understanding Color Models: A Review. ARPN Journal of Science and Technology, 2012. Vol. 2., No. 3., P. 265–275.
35. ISO/IEC 18004:2015 Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification. https://www.iso.org/standard/62021.html
36. ISO/IEC 9126-1:2001. Software Engineering – Product Quality. Part 1: Quality Model, 2001.
37. iSupply DMS – Distribution Management. https://lsi.net.au/solutions/isupply-dms-distribution-management
38. Ivan Mistrik et al. Relating System Quality and Software Architecture. 2014. 420 p.
39. Jan Bosch et al. Software Architecture: System Design, Development and Maintenance. Proceedings of 17th World Computer Congress, 2002. 242 p.
40. Jian-Ping Su et al. i-Logistics: An Intelligent Logistics System Based on Internet of Things. Proceedings of the IEEE International Conference on Applied System Innovation, 2017. P. 331–334.
41. Jureta I.J., Faulkner S., Schobbens P.-Y. A more expressive softgoal conceptualization for quality requirements analysis. Springer, 2006. Vol. 4215, P. 281–295.
42. Kaywa QR Code. https://qrcode.kaywa.com
43. KeepTruckin. https://keeptruckin.com/features
44. Kim Y. J., Lee J. Y. Algorithm of a Perspective Transform-Based PDF417 Barcode Recognition. Wireless Personal Communications, 2016. Vol. 89(3), P. 893–911. doi:10.1007/s11277-016-3171-6
45. Klumpp M., Zijm H. Logistics innovation and social sustainability: How to prevent an artificial divide in Human–Computer Interaction. Journal of Business Logistics. 2019. Vol. 40, Issue 3, P. 265–278. https://doi.org/10.1111/jbl.12198

46. Kuebix. https://www.kuebix.com
47. Leite J.C., Hadad G., Doorn J., Kaplan G. A Scenario Construction Process. Requirements Engineering Journal, 2000. Vol. 5(1), P. 38–61.
48. Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice. Addison-Wesley Professional, 2003. 528 p.
49. Lenberg P., Feldt R., Wallgren L. G. Behavioral software engineering: A definition and systematic literature review. Journal of Systems and software, 2015. Vol. 107, P.15–37.
50. Lin D.-T., Lin M.-C., Huang K.-Y. Real-time automatic recognition of omnidirectional multiple barcodes and DSP implementation, Mach. Vision Appl. 2011. Vol. 22(2), P. 409–419.
51. Logistics Software International. https://lsi.net.au/solutions
52. Lüder A., Schmidt N., Drath R. Standardized information exchange within production system engineering. Springer, Multi-Disciplinary Engineering for Cyber-Physical Production Systems, 2017. P. 235–257.
53. Maalej W., Nabil H. Bug report feature request or simply praise? On automatically classifying app reviews. Proceedings of the 23rd IEEE International Requirements Engineering Conference. IEEE, 2015. P. 116–125.
54. Maalej W., Nayebi M., Johann T., Ruhe G. Toward data-driven requirements engineering. IEEE Software, 2016. Vol. 33, Issue 1, P. 48–54.
55. Magaya Supply Chain. https://www.magaya.com/magaya-supply-chain
56. Mahmoud A., Grant W. Detecting classifying and tracing non-functional software requirements. Requirements Engineering, 2016. Vol. 21, Issue 3, P. 1–25.
57. Maintainability Index Range and Meaning. Microsoft, 2007. https://docs.microsoft.com/en-us/archive/blogs/codeanalysis/maintainability-index-range-and-meaning

58. Martin W., Sarro F., Jia Y., Zhang Y., Harman M. A Survey of app store analysis for software engineering. IEEE Transactions on Software Engineering, 2016.
59. Mary Shaw, David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996. 242 p.
60. Matthias Klumpp, Marc Hesenius, Ole Meyer, Caroline Ruiner, Volker Gruhn. Production logistics and human-computer interaction—state-of-the-art, challenges and requirements for the future. International Journal of Advanced Manufacturing Technology, 2019. Vol. 105, P. 3691–3709.
61. Mayvan B. B., Rasoolzadegan A., Yazdi Z. G. The state of the art on design patterns: A systematic mapping of the literature. Journal of Systems and Software, 2017. Vol. 125, P. 93–118.
62. McIlroy S., Ali N., Khalid H., Hassan A. E. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. Empirical Software Engineering, 2016. Vol. 21, Issue 3, P. 1067–1106.
63. McIlroy S., Shang W., Ali N., Hassan A. Is it worth responding to reviews? A case study of the top free apps in the Google Play store. IEEE Software, 2015.
64. Mehta D. P. Handbook of Data Structures and Applications. CRC Press, 2018. 1100 p.
65. Mengmeng Lu, Peng Liang. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, 2017. P. 344–353. doi.org/10.1145/3084226.3084241.
66. Neal Ford, Mark Richards. Fundamentals of Software Architecture: A Comprehensive Guide to Patterns, Characteristics, and Best Practices. O'Reilly Media. 2020. 500 p.
67. Ng W.W., Lin L., Chan P.P., Yeung D.S. 3D goods allocation in warehouse with L-GEM based 3-D RFID positioning. Proceedings of the International

Conference on Machine Learning and Cybernetics (ICMLC), 2011, Vol. 1, P. 324–329.

68. Nick Rozanski, Eóin Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Pearson Education, 2005. 576 p.
69. OMG® Unified Modeling Language®, Version 2.5.1. Object Management Group, 2017. 796 p.
70. Pagano D., Maalej W. User feedback in the appstore: an empirical study. Proceedings of the 21st IEEE International Requirements Engineering Conference, 2013. P. 125–134.
71. Pang J., Shen L., Zhang Q., Xu H., Li P. Design of Modern Logistics Management System Based on RFID and NB-IoT. Advances in Intelligent Systems and Computing, 2019. Vol 927. Springer, Cham. https://doi.org/10.1007/978-3-030-15035-8_54
72. Panichella S. Di, Sorbo A., Guzman E., Visaggio C. A., Canfora G., Gall H. C. How can I improve my app? Classifying user reviews for software maintenance and evolution. Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution. 2015, P. 281–290.
73. Petriashvili Lili, Zhvania Taliko, Kapanadze David. Process Management in Warehousing Logistics by Means of RFID Automated System. Journal of Multidisciplinary Engineering Science Studies (JMESS), 2017. Vol. 3, Issue 3, P. 1502– 1505.
74. Pfeiffer T., Hellmers J., Schön E. M., Thomaschewski J. Empowering user interfaces for Industrie 4.0. Proceedings of the IEEE, 2016. Vol. 104, Issue 5, P. 986–996.
75. QR Code Generator. https://qr-code-generator.com
76. QR Code Maker. https://play.google.com/store/apps/details?id= com.qrcodemaker

77. QR Code Reader. https://play.google.com/store/apps/details?id=com.betteridea.barcode.qrcode
78. QR Code. Synthesis Journal, 2008. P. 59–78.
79. QR Creator. https://play.google.com/store/apps/details?id=com.qrcreator.meteorrain
80. QR scanner. https://play.google.com/store/apps/details?id=app.qr.qrcode.qrscanner
81. QR-Code Generator. https://play.google.com/store/apps/details?id=com.ykart.tool.qrcodegen
82. QR-Code Studio. https://www.tec-it.com/ru/download/free-software/qrcode-studio/Download.aspx
83. Querini M., Grillo A., Lentini A., Italiano G.F. 2D Color Barcodes for Mobile Phones. International Journal of Computer Science and Applications, 2011. Vol. 8, No. 1. P. 136–155.
84. Querini M., Italiano G.F. Color Classifiers for 2D Color Barcodes. Proceedings of the Federated Conference onComputer Science and Information Systems. 2013. P. 611–618.
85. Querini M., Italiano G.F. Reliability and Data Density in High Capacity ColorBarcodes. Computer Science and Information System, 2014. Vol. 11, No. 4. P.1595–1615.
86. Rackbeat. Warehouse Management System. https://rackbeat.com
87. Raghvinder S. Sangwan. Software and Systems Architecture in Action. CRC Press, 2014. 232 p.
88. Riaz M. N. Impact of software design patterns on the quality of software: A comparative study. International Conference on Computing, Mathematics and Engineering Technologies, 2018. P. 1–6.
89. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. ADDISON-WESLEY, 1999. 568 p.

90. Schiffer Maximilian, Schneider Michael, Laporte Gilbert. Designing sustainable mid-haul logistics networks with intra-route multi-resource facilities, European Journal of Operational Research, 2018. Vol. 265, Issue 2, P. 517–532, doi.org/10.1016/j.ejor.2017.07.067.
91. Saher N., Baharom F., Romli R. A Review of Requirement Prioritization Techniques in Agile Software Development. Proceedings of the Knowledge Management International Conference (KMICe), 2018. P. 242–247.
92. Sik-Lányi C. Choosing effective colours for websites. Colour Design: Theories and Applications. Woodhead Publishing Series in Textiles, 2012. P. 600–621
93. Softeon. https://www.softeon.com/our-solutions
94. Technical Guideline TR-03137. Optically Verifiable Cryptographic Protection of non-electronic Documents (Digital Seal). 2020. 64 p.
95. TEC-IT QR Code Generator. https://qrcode.tec-it.com
96. The Science of Color Contrast – An Expert Designer's Guide. https://medium.muz.li/the-science-of-color-contrast-an-expert-designers-guide-33e84c41d156
97. Transforming the Last Mile. The Global Language of Business. 2020. https://www.gs1.org
98. Uniform symbology specification PDF417. AIM USA, 2009. http://www.aimusa.org.
99. Unitag. https://www.unitag.io/qrcode
100. Van Lamsweerde, A. Goal-Oriented Requirements Engineering: A Guided Tour. Proceedings of the 5th IEEE international Symposium on Requirements Engineering, 2001, P. 249.
101. Vu P. M., Nguyen T. T., Pham H. V. Mining user opinions in mobile app reviews: a keyword-based approach. Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, 2015. P. 749–759.

102. Wachenfeld S., Terlunen S., X Jiang. Robust 1-D barcode recognition on camera phones and mobile product information display, Mobile Multimedia Processing, Springer, 2010. P. 53–69.
103. Web Content Accessibility Guidelines (WCAG) 2.0, https://www.w3.org/TR/2008/REC-WCAG20-20081211
104. Wedyan F., Abufakher S. Impact of design patterns on software quality: a systematic literature review. IET Software, 2019. Vol. 14, Issue 1, P.1–17.
105. Welch T. A Technique for High-Performance Data Compression. Computer, 1984. Vol. 17, Issue 6, P. 8–19.
106. Winkelhaus Sven, Grosse Eric H. Logistics 4.0: a systematic review towards a new logistics system. International Journal of Production Research, 2020. Vol. 58, Issue 1, P. 18–43. DOI:10.1080/00207543.2019.1612964
107. Xinwen Zhang et al. LIPPS: Logistics Information Privacy Protection System based on Encrypted QR Code. IEEE IEEE TrustCom-BigDataSE-ISPA, 2016. P. 996–1000.
108. Yang H., Chen L., Chen Y., Lee Y., Yin Z. Automatic barcode recognition method based on adaptive edge detection and a mapping model. Journal of Electronic Imaging, 2016. Vol. 25(5), 053019. doi:10.1117/1.jei.25.5.053019
109. Yang L., Chen Y., Li X.Y., Xiao C., Li M., Liu Y. Tagoram: real-time tracking of mobile RFID tags to high precision using cots devices. Proceedings of the 20th Annual International Conference on Mobile Computing and Networking, 2014, P. 237–248.
110. Young Jung Kim, Jong Yun Lee. Algorithm of a Perspective Transform-Based PDF417 Barcode Recognition. Wireless Pers Communication. Springer, 2016. 19 p. DOI 10.1007/s11277-016-3171-6
111. Zhu S., Song J., Hazen B.T., Lee K., Cegielski C. How supply chain analytics enables operational supply chain transparency: an organizational

information processing theory perspective. Int. Journal of Phys. Distrib. Logist. Manag., 2018. Vol. 48(1). P. 47–68.

112. Zijm W.M.H., Klumpp M. Logistics and supply chain management: trends and developments. Logistics and supply chain innovation. 2016. Springer, Berlin, P. 1–20.

113. Бойко Н. І. Інформаційна логістика підприємства. Вісник Національного університету «Львівська політехніка», 2007. № 580. С. 3–9.

114. Бойко Н. І. Логістичний менеджмент торговельного підприємства (на прикладі мережі супермаркетів «Сільпо»). Вісник Національного університету «Львівська політехніка», 2007. № 594. С. 183–189.

115. Дичка І.А., Зорін Ю.М., Онай М.В. Особливості подання алфавітно-цифрових даних у графічно-кодованому вигляді. Наук. вісник Чернівецького університету. 2008. № 426. С. 145–152.

116. Дичка І.А., Голуб В.І., Новосад М.В. Графічне кодування інформації в поштовій галузі. Радіоелектроніка і інформатика, 2010. № 1. С. 79–83.

117. Дичка І. А. Забезпечення завадостійкості багатокольорових двомірних штрихових кодів. Наук. вісті НТУУ «КПІ». 2003. № 1 (27). С. 38–43.

118. Дичка І. А. Завадостійкі лінійні штрихові коди з виправленням асиметричних помилок у штрихкодових знаках. Наук. вісті НТУУ «КПІ». 2002. № 6 (26). С. 41–49.

119. Дичка І. А. Методика вибору та проектування лінійних штрихових кодів. Наук. вісті НТУУ «КПІ». 2002. № 5 (25). С. 35–40.

120. Дичка І. А. Перетворення інформації при її поданні у вигляді багатокольорових матричних штрихових кодів. Наук. вісті НТУУ «КПІ». 2003. № 2 (28). С. 49–55.

121. Дичка І. А. Підвищення завадостійкості лінійних штрихових кодів. Вісн. Технол. ун-ту Поділля. 2002. Вип. 3, т.2. С. 78–86.

122. Дичка І. А. Побудова високощільних кольорових лінійних штрихових кодів. Наук. вісті НТУУ «КПІ». 2002. № 2 (22). С. 29–34.
123. Дичка І. А. Побудова кольорових лінійних штрихових кодів. Наук. вісті НТУУ «КПІ». 2002. № 1 (21). С. 20–25.
124. Дичка І. А. Побудова неструктурованих лінійних штрихових кодів. Наук. вісті НТУУ «КПІ». 2002. № 3 (23). С. 32–37.
125. Дичка І. А. Порівняння лінійних чорно-білих та кольорових штрихових кодів. Вісн. Житомирського інж.-технол. ін-ту. Житомир: ЖІТІ, 2002. Вип. 1 (20). С. 75–78.
126. Дичка І. А. Структурні способи підвищення інформаційної щільності лінійних штрихових кодів. Наук. вісті НТУУ «КПІ». 2002. № 4 (24). С. 13–19.
127. Дичка І. А., Голуб В. В., Каплинський П. М. Обробка двомірних багатокольорових штрихових кодів. Вісн. Технол. ун-ту Поділля. 2003. Вип. 3, т. 1. С. 82–89.
128. Дичка І. А., Новосад М. В. Забезпечення завадостійкості інформації, що зберігається у вигляді графічного коду. Вісник КрНУ імені Михайла Остроградського. 2011. Випуск 6 (71). Частина 1. С. 42–48.
129. Дичка І. А., Новосад М. В. Оцінювання ступеня ущільнення алфавітно-цифрових даних при їх поданні у графічно-кодованому вигляді. Вісник Хмельницького національного університету. 2011. № 4. С. 200–207.
130. Дичка І.А. Зберігання інформації у вигляді багатокольорових штрихових кодів та їх обробка. К. : ІВЦ “Видавництво “Політехніка“, 2003. 340 с.
131. Дичка І.А., Голуб В.І., Новосад М.В. Метод ущільнення алфавітно-цифрової інформації, поданої в графічнокодованому вигляді. Наукові вісті НТУУ «КПІ». 2011. № 2. С. 69–76.

132. Дичка І.А., Новосад М.В., Грибок Т.Ю. Преобразование информации при создании и обработке многоцветных графических кодов. Известия вузов. Радиоэлектроника. 2013. № 7. С. 18–28.
133. Дичка І.А., Онай М.В., Новосад М.В. Подання інформації у графічно-кодованому вигляді: технологія кодування та декодування даних. Реєстрація, зберігання і обробка даних, 2010. Т. 12, № 2. С. 69–80.
134. ДСТУ 3146-95. Коди та кодування інформації. Штрихове кодування. Маркування об`єктів ідентифікації. Штрихкодові позначки EAN. https://zakon.rada.gov.ua/rada/show/n0005699-96#Text
135. Жураковський Б. Ю. Використання методів адаптивного кодування для каналів з параметрами, що змінюються. Вісник ДУІКТ, 2007. № 5(2). С. 199–202.
136. Жураковський Б. Ю. Порівняльний аналіз формування та застосування двомірних штрих-кодів для передачі даних. Системи управління, навігації та зв'язку, 2015. № 2 (34), С.68–70.
137. Жураковський Б. Ю. Сфери застосування двовимірних штрихових кодів. Системи управління, навігації та зв'язку, 2016. № 2(38), С.83–87.
138. Жураковський Б. Ю., Дружинін В. А. Багатовимірні штрихові коди. Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління», 2018. № 2 (33). С. 15–31.
139. Кислий В. М., Біловодська О. А., Олефіренко О. М., Соляник О. М. Логістика: Теорія та практика: Навч. посіб. К : Центр учбової літератури, 2010. 360 с.
140. Система GS1. Ідентифікація та штрихове кодування. https://gs1ua.org/ua/gs1-system/identifikatsiya-ta-shtrihove-koduvannya/about-barcodes

141. Скубак О. М., Мокринцев О. А. Технологія кодування сучасних штрих-кодів. Телекомунікаційні та інформаційні технології. 2016. №1. С. 74–79.

142. Тарасенко В. П., Дичка І. А. Виправлення двократних помилок у штрихкодових знаках кольорових лінійних штрихових кодів. Вимірювальна та обчислювальна техніка в технологічних процесах. 2001. № 4. С. 187–193.

143. Тарасенко В. П., Дичка І. А. Зберігання даних у вигляді кольорових матричних штрихових кодів та їх обробка. Автоматизированные системы управления и приборы автоматики. 2002. Вып. 121. С. 9–15.

144. Тарасенко В. П., Дичка І. А. Корекція помилок у кольорових стекових штрихових кодах. Інформаційно-керуючі системи на залізничному транспорті. 2002. № 6. С. 89–93.

145. Тарасенко В. П., Дичка І. А. Побудова завадостійких багатокольорових матричних штрихових кодів. Радиоэлектроника и информатика. 2003. № 2(23). С. 89–93.

146. Тарасенко В. П., Дичка І. А. Побудова кольорових матричних штрихових кодів з корекцією помилок. Радиоэлектроника и информатика. 2002. № 4(21). С. 106–108.

147. Тарасенко В. П., Дичка І. А. Побудова кольорових стекових штрихових кодів. Автоматизація виробничих процесів. 2002. № 2 (15). С. 26–36.

148. Тарасенко В. П., Дичка І. А. Проектування кольорових лінійних штрихових кодів з корекцією помилок у штрихкодових знаках. Вісн. Технол. ун-ту Поділля. 2002. Вип. 5, ч.1. С. 125–131.

149. Тарасенко В. П., Дичка І. А. Способи ущільнення даних при їх поданні у вигляді кольорових стекових штрихових кодів. Вісн. Технол. ун-ту Поділля. 2002. Вип. 6, ч.1. С. 256–263.

150. Тарасенко В. П., Дичка І. А. Структурні способи підвищення надійності зберігання даних у вигляді багатокольорових матричних штрихових кодів. Автоматизированные системы управления и приборы автоматики. Х., ХНУРЭ, 2003. Вып. 123. С. 101–109.
151. Швець О. Занурення в патерни проектування. Refactoring.Guru, 2019. 396 с.

ДОДАТКИ

Додаток А. Будова ШК-позначки та порівняння кодів EAN

Рис. А.1. Специфікація позначки коду EAN-13 [19]

(а) (б)

Рис. А.2. Приклад позначки коду EAN-13: (а) з доданим кодом EAN-5; (б) з доданим кодом EAN-2 [2]

(а) (б)

Рис. А.3. Приклад позначки коду UPC: (а) UPC-A; (б) UPC-E [2]

Рис. А.4. Приклад позначки коду GS1-128 [140]

Таблиця А.1. Порівняльна таблиця кодів сімейства EAN / UPC

Назва	Набір даних	Об’єм даних*	Визначення контрольного значення	Особливості
EAN-2	Цифри	2 байти	Відсутнє контрольне значення	Використовується разом з EAN-13, EAN-8 та UPC.
EAN-5	Цифри	5 байт	Відсутнє контрольне значення	Використовується разом з EAN-13, EAN-8 та UPC.
EAN-8	Цифри	7 байт	1 контрольне значення	–
EAN-13	Цифри	12 байт	1 контрольне значення	–
EAN-14	Цифри	13 байт	1 контрольне значення	Містить додатковий префікс «01».
EAN-18	Цифри	17 байт	1 контрольне значення	Містить додатковий префікс «00».
EAN-99	Цифри	10 байт	1 контрольне значення	Містить префікс «99».
EAN-128	Символи ASCII	Довільний	1 контрольне значення	Використовуються 3 набори**
ISBN	Цифри	9 байт	1 контрольне значення	Містить префікс «978» або «979».
ISSN	Цифри та «X»	4 байти	1 контрольне значення	Містить префікс «977».
UPC-A	Цифри	11 байт	1 контрольне значення	–
UPC-E	Цифри	6 байт	1 контрольне значення	Містить префікс «0».

Примітки. * Об’єм даних наведено без врахування контрольного значення та/або префіксу.
** Можуть використовуватися символи, що належать до одного з трьох наборів («A», «B», «C»), про що визначається спеціальним префіксним символом «Start A», «Start B», «Start C» відповідно.

Додаток Б. Будова ШК-позначки та характеристики QR-коду

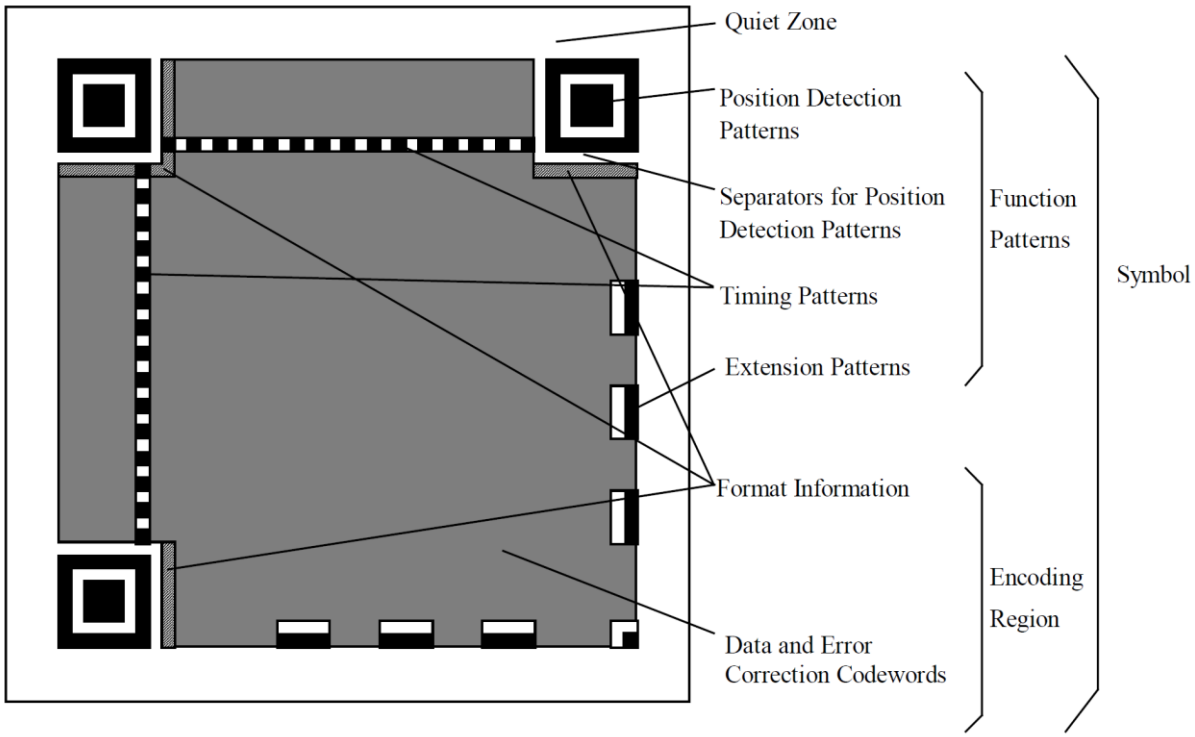


Рис. Б.1. Функціональні елементи позначки QR-коду [35]

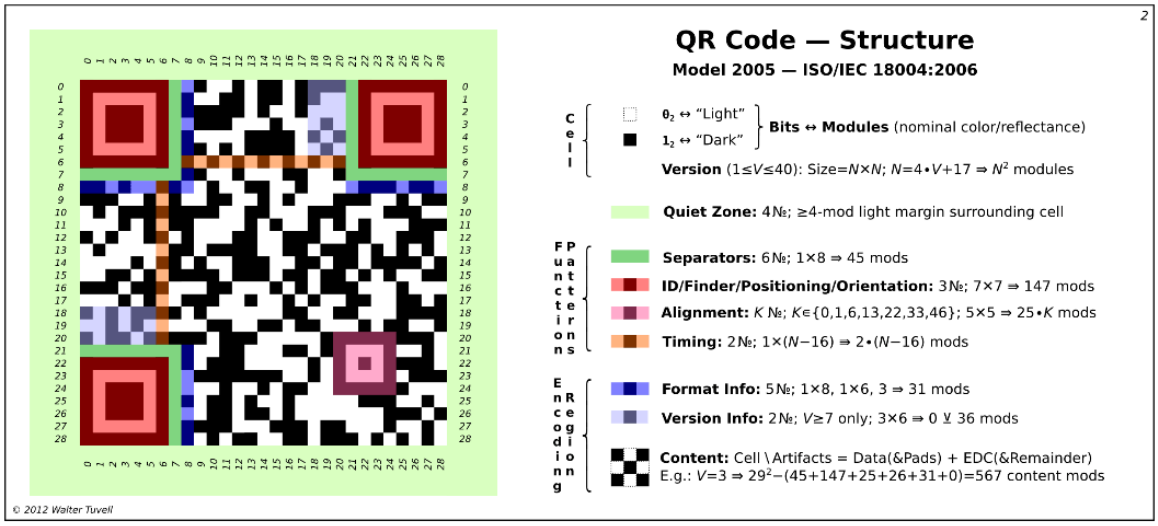


Рис. Б.2. Загальна будова позначки QR-коду [78]

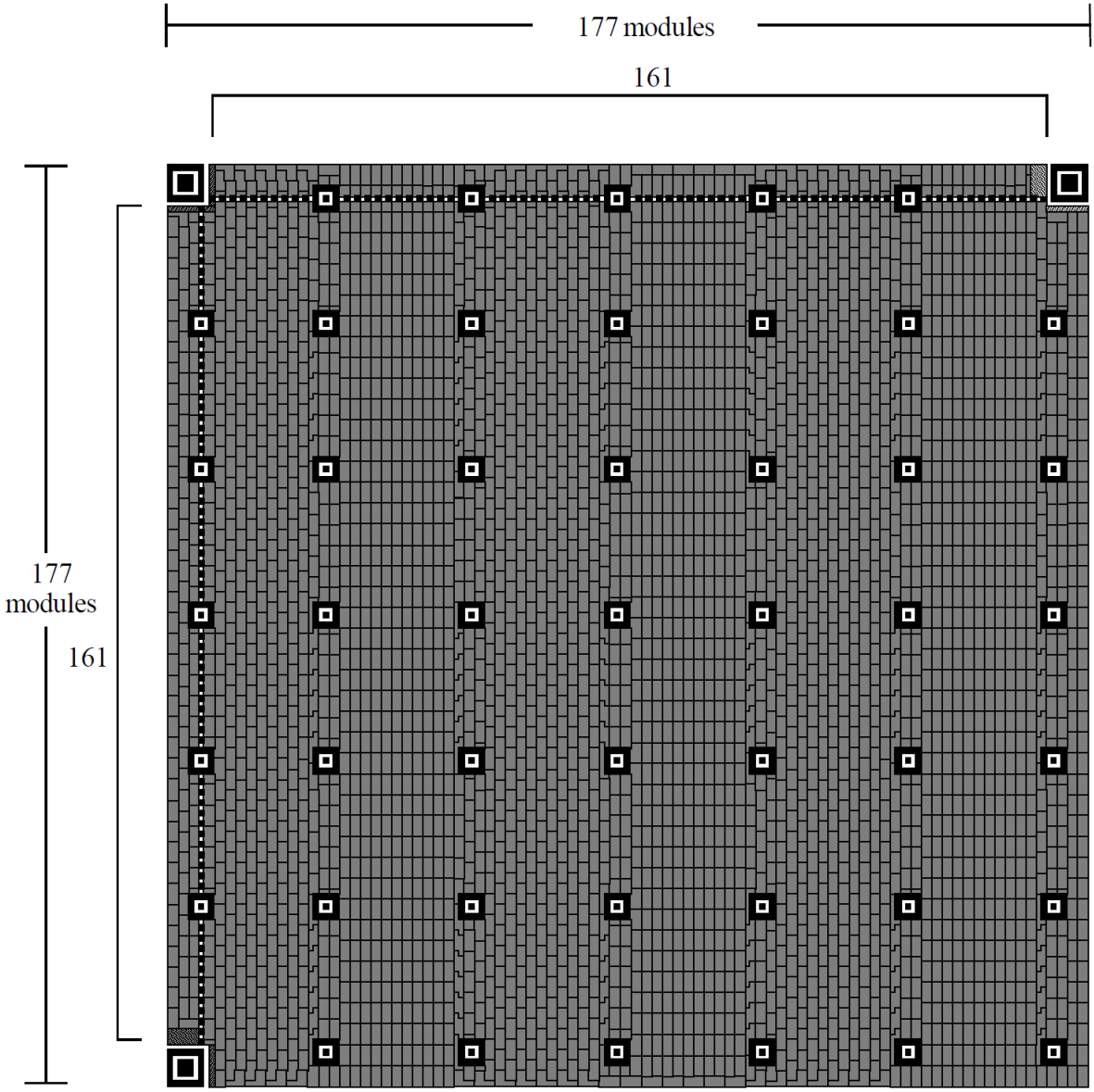


Рис. Б.3. Структура позначки QR-коду версії 40 [35]

Таблиця Б.1. Залежність максимальної ємності позначки від типу даних

Тип даних	Максимальна ємність позначки	Комірок/символ
Цифри	7089 цифр	3,3
Алфавітно-цифрові символи	4296 символів	5,5
Бінарні значення	2953 значень	8
Символи системи запису Кандзі	1817 символів	13

Таблиця Б.2. Завадостійкість QR-коду версії 40

Назва рівня	Кількість кодових слів для виправлення помилок	Кількість кодових слів даних	Кількість блоків виправлення помилок	Виправлення помилок у блоці, (c, k, r)
Level L	750	2956	19	(148,118,15)
			6	(149,119,15)
Level M	1372	2334	18	(75,47,14)
			31	(76,48,14)
Level Q	2040	1666	34	(54,24,15)
			34	(55,25,15)
Level H	2430	1276	20	(45,15,15)
			61	(46,16,15)

Таблиця Б.3. Залежність інформаційної ємності від рівня завадостійкості для QR-коду версії 40

Назва рівня	Об’єм даних, біти	Кількість символів			
		Цифри	Алфавітно-цифрові символи	Бітові значення	Символи системи Кандзі
Level L	23 648	7 089	4 296	2 953	1 817
Level M	18 672	5 596	3 391	2 331	1 435
Level Q	13 328	3 993	2 420	1 663	1 024
Level H	10 208	3 057	1 852	1 273	784

Таблиця 1.6. Порівняльний аналіз програмних застосунків для роботи з позначками QR-коду

Назва	Тип	Максимальна кількість символів при формуванні позначки	Можливість розпізнавання тексту максимального обсягу
QR Code Generator [75]	Веб-сервіс	Близько 950	Розпізнає

Unitag [99]	Веб-сервіс	Близько 7000	Створює і кодує посилання на текст
TEC-IT QR Code Gen [95]	Веб-сервіс	Близько 2930	Розпізнає
Kaywa QR Code [42]	Веб-сервіс	160	Розпізнає
Create QR [12]	Веб-сервіс	200	Розпізнає
QR-Code Studio [82]	Комп’ютерний застосунок	2953	Не розпізнає
CodeTwo QR Code Desktop Reader & Generator [11]	Комп’ютерний застосунок	1024	Не завжди
QR Code Reader [77]	Мобільний застосунок	500	Розпізнає
QR-Code Generator [81]	Мобільний застосунок	300	Розпізнає
QR Creator [79]	Мобільний застосунок	Близько 2800	Застосунок розпізнає, стороннє ПЗ не розпізнає
QRCode Scanner [80]	Мобільний застосунок	1000	Розпізнає
QR Code Maker [76]	Мобільний застосунок	Близько 2800-2900	Не завжди

Додаток В. Будова та характеристики коду JAB

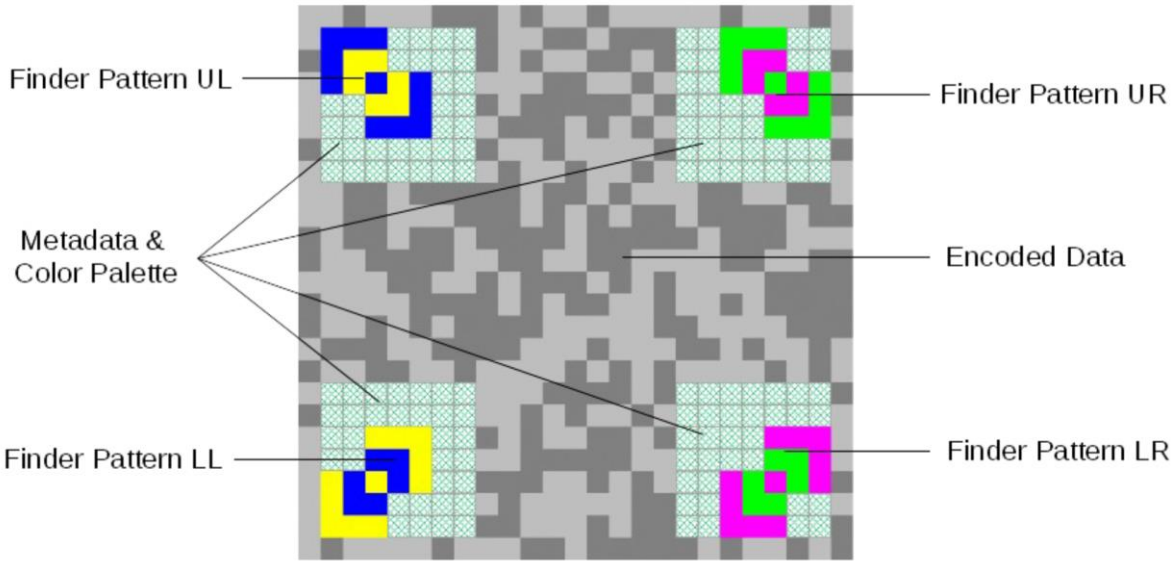


Рис. В.1. Структура первинного символу [94]

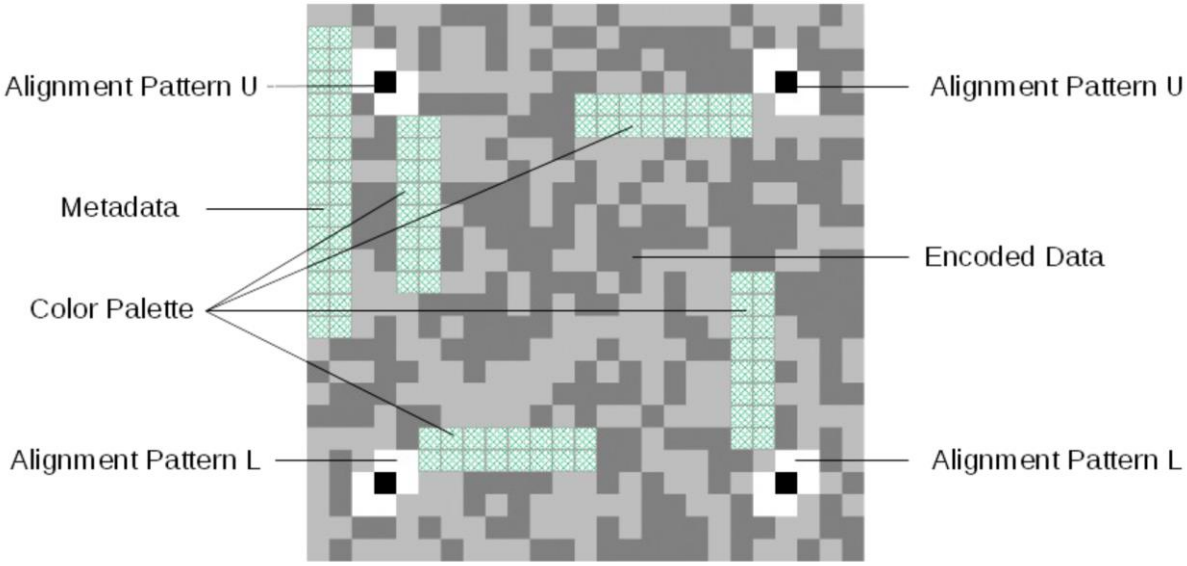


Рис. В.2. Структура вторинного символу [94]

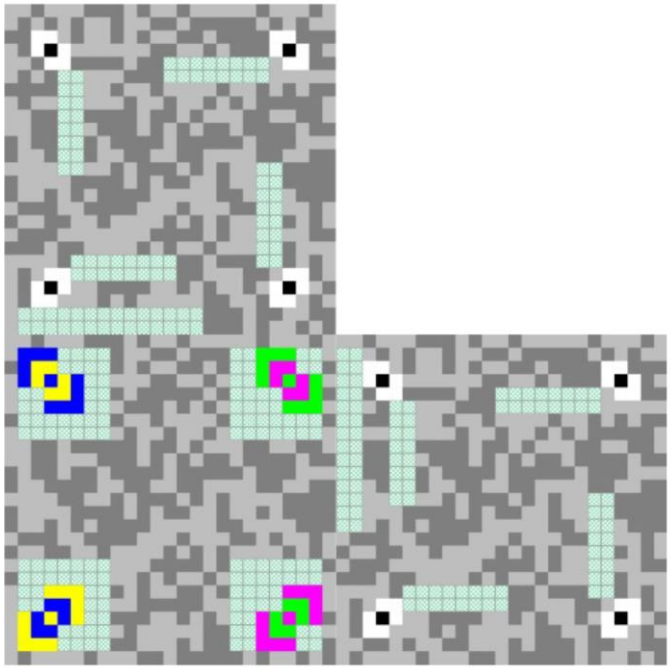


Рис. В.3. Приклад комбінування одного первинного та двох вторинних символів [94]

Таблиця В.1. Порівняння кольорових матричних кодів

Код	Форма комірки	Кількість кольорів	Форма позначки	Інформаційна ємність, біти	Забезпечення завадостійкості
JAB	Квадратна	Від 4 до 256	Квадратна або прямокутна	Від 312 до 27000	За рахунок коду LDPC
HCCB	Трикутна	4 або 8	Квадратна	160	За рахунок алгоритму RSA-1024
HCC2D	Квадратна	4, 8 або 16	Квадратна	Від 416 до 6272	За рахунок коду Ріда-Соломона

Таблиця В.2. Інформаційна ємність квадратної позначки коду JAB

Розмір сторони (у модулях)	Інформаційна ємність, біти			
	Первинна позначка		Вторинна позначка	
	4 кольори	8 кольорів	4 кольори	8 кольорів

21	312	466	362	534
25	469	703	520	771
33	867	1299	918	1368
49	1990	2985	2042	3054
57	2714	4073	2769	4145
61	3119	4680	3174	4752
69	4010	6017	4065	6089
77	5011	7518	5066	7590
85	6120	9182	6177	9257
97	7992	11990	8049	12065
101	8670	13008	8728	13083
109	10110	15168	10168	15243
113	10872	16310	10929	16385
121	12476	18717	12534	18792
129	14190	21288	14248	21363
137	16014	24024	16072	24099
141	16968	25454	17025	25529
145	17948	26925	18006	27000

Таблиця В.3. Завадостійкість позначки для різних рівнів

Рівень завадостійкості	Відновлювальна здатність, %	Швидкість коду
0	3	0.67
1	4	0.63
2	5	0.57
3	6	0.55
4	7	0.50
5	8	0.43
6	9	0.34
7	10	0.25
8	11	0.20
9	12	0.17
10	14	0.14

Додаток Г. Ущільнення даних при формуванні штрихкодових позначок

Таблиця Г.1. Розв'язки системи нерівностей (2.7) для $s = 4$

n	m	Потужність алфавіту A	Коефіцієнт ущільнення $U^{(4)}_{P_{\Omega_{inf}}}$
5	3	10	1,250
8	5	10	1,200
9	5	10	1,350
11	6	10	1,375
11	8	10	1,031
12	7	10	1,286
13	8	10	1,219
14	9	10	1,167
16	9	10	1,333
17	10	10	1,275
5	3	11	1,250
7	5	11	1,050
10	7	11	1,071
11	8	11	1,031
12	7	11	1,286
13	8	11	1,219
14	9	11	1,167
17	10	11	1,275
5	3	12	1,250
8	5	12	1,200
11	7	12	1,179
13	8	12	1,219
14	9	12	1,167
17	10	12	1,275
7	5	13	1,050
10	7	13	1,071
11	8	13	1,031
13	8	13	1,219
14	9	13	1,167
7	5	14	1,050
10	7	14	1,071
11	8	14	1,031
14	9	14	1,167

7	5	15	1,050
11	8	15	1,031
14	9	15	1,167
7	5	16	1,050
11	8	16	1,031
7	5	17	1,050
11	8	17	1,031
7	5	18	1,050
11	8	18	1,031
7	5	19	1,050
11	8	19	1,031
11	8	20	1,031
5	4	28	1,250
7	6	28	1,167
9	8	28	1,125
11	9	28	1,222
5	4	29	1,250
7	6	29	1,167
9	8	29	1,125
11	9	29	1,222
6	5	30	1,200
8	7	30	1,143
10	9	30	1,111
11	10	30	1,100
7	6	31	1,167
9	8	31	1,125
11	9	31	1,222
6	5	32	1,200
8	7	32	1,143
10	9	32	1,111
11	10	32	1,100
7	6	33	1,167
9	8	33	1,125
11	10	33	1,100
7	6	34	1,167
9	8	34	1,125
11	10	34	1,100
8	7	35	1,143
10	9	35	1,111

7	6	36	1,167
9	8	36	1,125
11	10	36	1,100
8	7	37	1,143
10	9	37	1,111
7	6	38	1,167
9	8	38	1,125
11	10	38	1,100
9	8	39	1,125
11	10	39	1,100
9	8	40	1,125
11	10	40	1,100
9	8	41	1,125
11	10	41	1,100
10	9	42	1,111
9	8	43	1,125
11	10	43	1,100
11	10	44	1,100
11	10	45	1,100
11	10	47	1,100

Таблиця Г.2. Деякі цілочислові розв’язки системи (2.7) за $s = 6$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$	Коефіцієнт ущільнення $U_{718}^{(6)}(P_A)$
2	5	13	$5 \to 2$	1,250
3	4	138	$4 \to 3$	1,111
3	7	16	$7 \to 3$	1,167
4	7	42	$7 \to 4$	1,167
4	11	10	$11 \to 4$	1,375
5	7	109	$7 \to 5$	1,167
5	9	38	$9 \to 5$	1,200
6	7	280	$7 \to 6$	1,167
6	11	36	$11 \to 6$	1,222
6	17	10	$17 \to 6$	1,417
7	8	300	$8 \to 7$	1,143
7	10	99	$10 \to 7$	1,190

7	12	46	$12 \to 7$	1,143
8	15	33	$15 \to 8$	1,250
9	13	94	$13 \to 9$	1,204
9	16	40	$16 \to 9$	1,185
10	19	31	$19 \to 10$	1,267

Таблиця Г.3. Розв’язки системи нерівностей (2.7) для $s = 8$

n	m	Потужність алфавіту A	Коефіцієнт ущільнення $U^{(4)}_{P_{\Omega_{inf}}}$
1	9	2	1,125
1	10	2	1,250
1	11	2	1,375
1	12	2	1,500
2	18	2	1,125
2	19	2	1,188
2	20	2	1,250
2	3	300	1,125
1	6	4	1,500
2	11	4	1,375
2	12	4	1,500
3	17	4	1,417
3	18	4	1,500
3	19	4	1,583
1	5	5	1,250
2	10	5	1,250
3	15	5	1,250
3	16	5	1,333
4	20	5	1,250
3	14	6	1,167
4	19	6	1,188
2	9	7	1,125
4	18	7	1,125
3	11	10	1,375
4	15	10	1,406
5	19	10	1,425
5	18	11	1,350
2	7	12	1,313
4	14	12	1,313

3	10	13	1,250
4	6	300	1,125
5	17	13	1,275
6	20	13	1,250
4	13	14	1,219
5	16	15	1,200
6	19	16	1,188
1	3	18	1,125
2	6	18	1,125
3	9	18	1,125
4	12	18	1,125
5	15	18	1,125
6	18	18	1,125
5	13	29	1,300
7	18	30	1,286
2	5	33	1,250
4	10	33	1,250
6	15	33	1,250
8	20	33	1,250
7	17	37	1,214
5	12	38	1,200
8	19	40	1,188
6	9	300	1,125
3	7	43	1,167
6	14	43	1,167
7	16	46	1,143
4	9	49	1,125
8	18	49	1,125
9	20	52	1,111
5	11	54	1,100
8	12	300	1,125
10	19	101	1,188
9	17	104	1,181
8	15	108	1,172
7	13	113	1,161
6	11	120	1,146
5	9	131	1,125
10	18	131	1,125
9	16	140	1,111

9	14	283	1,167
10	15	300	1,125
7	11	268	1,179

Таблиця Г.4. Деякі цілочислові розв'язки системи (2.10) за $s = 4$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$	Коефіцієнт ущільнення $U_{70}^{(4)}(P_A)$
3	5	10	$5 \to 3$	1,250
5	8	10	$8 \to 5$	1,200
5	9	10	$9 \to 5$	1,350
6	11	10	$11 \to 6$	1,375
7	12	10	$12 \to 7$	1,286
8	13	10	$13 \to 8$	1,219
9	14	10	$14 \to 9$	1,167
9	16	10	$16 \to 9$	1,333
10	17	10	$17 \to 10$	1,275
2	3	16	$2 \to 3$	1,125
7	8	37	$8 \to 7$	1,143
9	10	37	$10 \to 9$	1,111
6	7	38	$7 \to 6$	1,167
8	9	38	$9 \to 8$	1,125
10	11	38	$11 \to 10$	1,100

Таблиця Г.5. Деякі цілочислові розв'язки системи (2.7) за $s = 5$

m	n	Потужність алфавіту P_A	Тип перетворення $n \to m$	Коефіцієнт ущільнення $U_{232}^{(5)}(P_A)$
2	3	37	$3 \to 2$	1,200
3	7	10	$7 \to 3$	1,400
4	9	11	$9 \to 4$	1,350
5	6	93	$6 \to 5$	1,200
5	8	30	$8 \to 5$	1,280
6	7	106	$7 \to 6$	1,167
7	8	117	$8 \to 7$	1,143

7	11	32	$11 \rightarrow 7$	1,257
7	15	12	$15 \rightarrow 7$	1,286
8	9	126	$9 \rightarrow 8$	1,125
13	13	28	$13 \rightarrow 8$	1,300
9	10	134	$10 \rightarrow 9$	1,111
9	11	86	$11 \rightarrow 9$	1,222
9	10	128	$10 \rightarrow 9$	1,111

Таблиця Г.6. Найбільш доцільні значення алфавіту P_A за $s = 5$

Потужність алфавіту P_A	Тип перетворення $n \rightarrow m$	Коефіцієнт ущільнення $U_{232}^{(5)}(P_A)$	Коментарі
10	$7 \rightarrow 3$	1,400	Доцільно використовувати для подання десяткових цифр
28	$8 \rightarrow 5$	1,280	Доцільно використовувати для подання шістнадцяткових цифр та певних спеціальних символів
82	$6 \rightarrow 5$	1,200	Доцільно використовувати для подання латинських літер, знаків пунктуації та певних спеціальних символів
128	$10 \rightarrow 9$	1,111	Доцільно використовувати для подання базового алфавіту ASCII

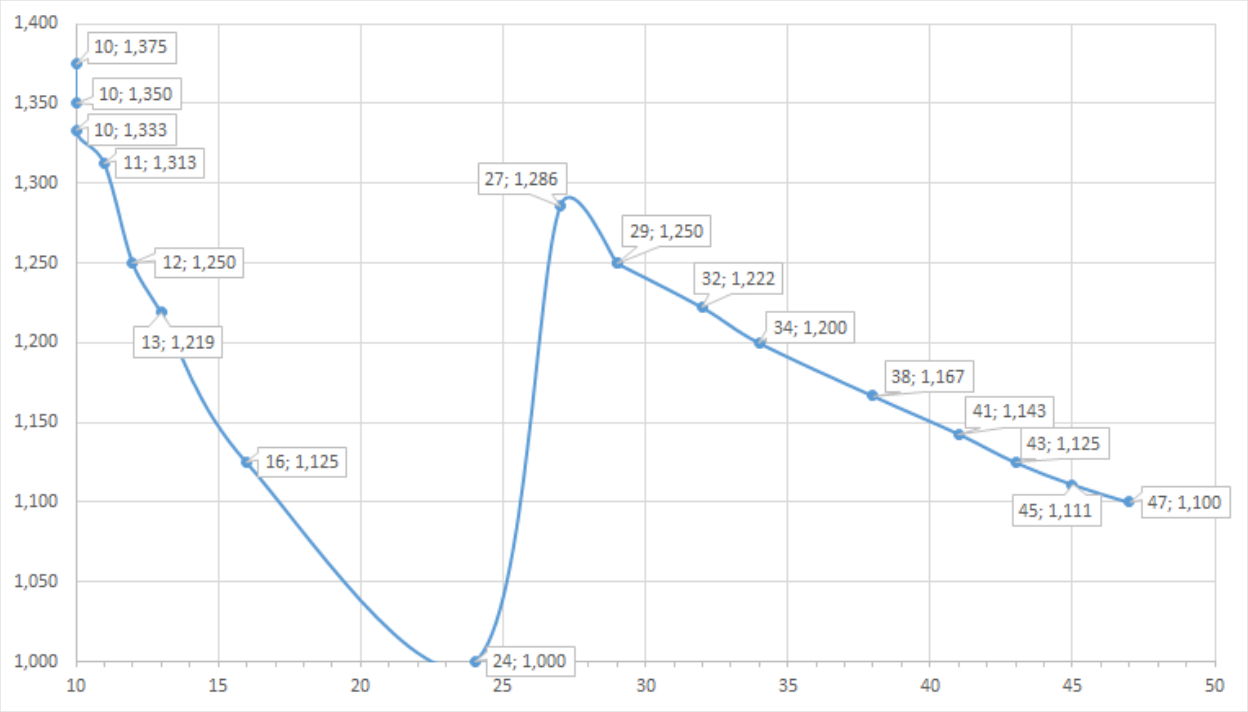


Рис. Г.1. Залежність коефіцієнту ущільнення від потужності алфавіту за $s = 4$

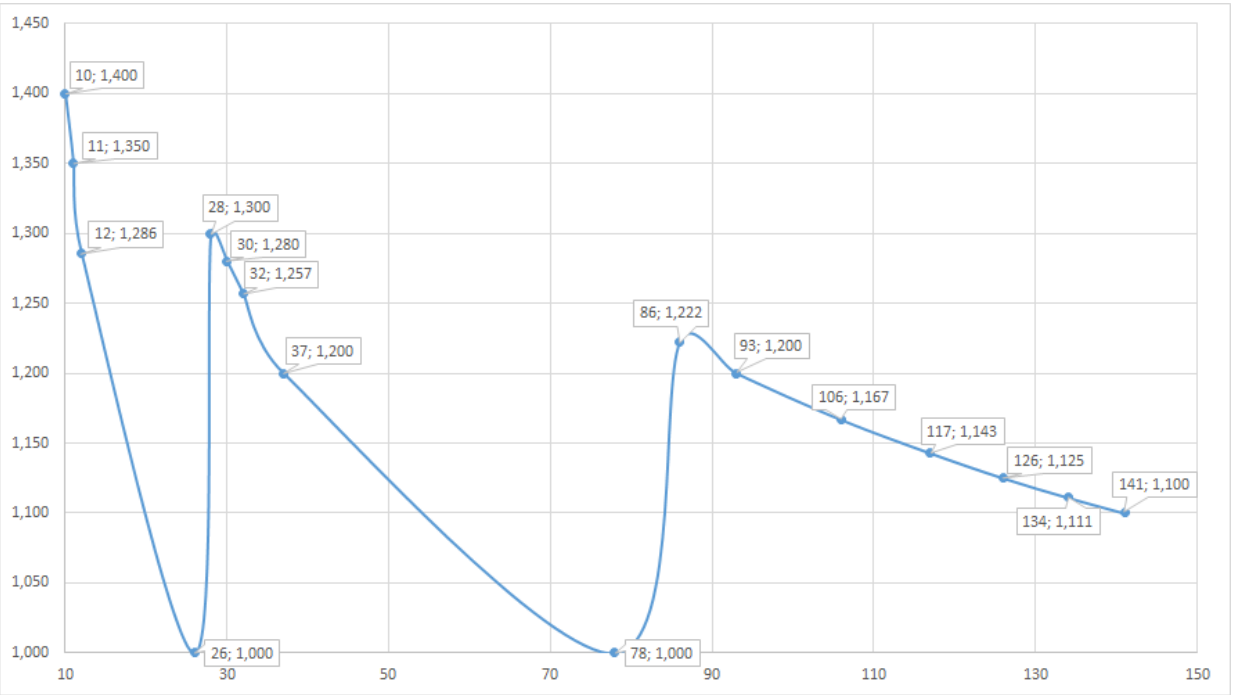
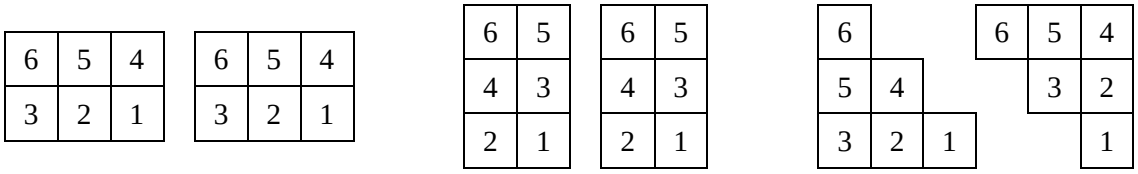


Рис. Г.2. Залежність коефіцієнту ущільнення від потужності алфавіту за $s = 5$



(а) (б) (в)

Рис. Г.3. Можливі конфігурації стикування ШК-знаків залежно від їхньої форми

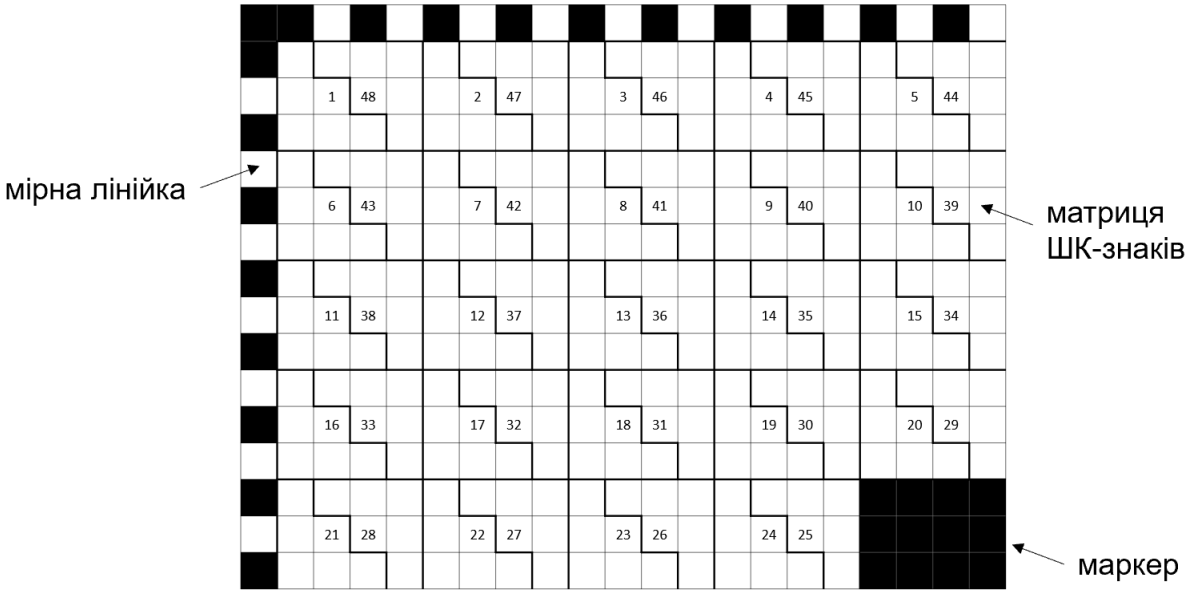
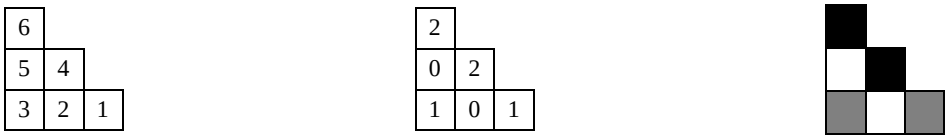


Рис. Г.4. Структура ДШК-позначки



Порядок розрядів у ШК-знаку (6 – старший розряд, 1 − молодший)

Трійкова карта ШК-знаку, що відповідає кодовектору 101202

Графічне зображення ШК-знаку на носії

Рис. Г.5. Приклад ШК-знаку за $s = 6$

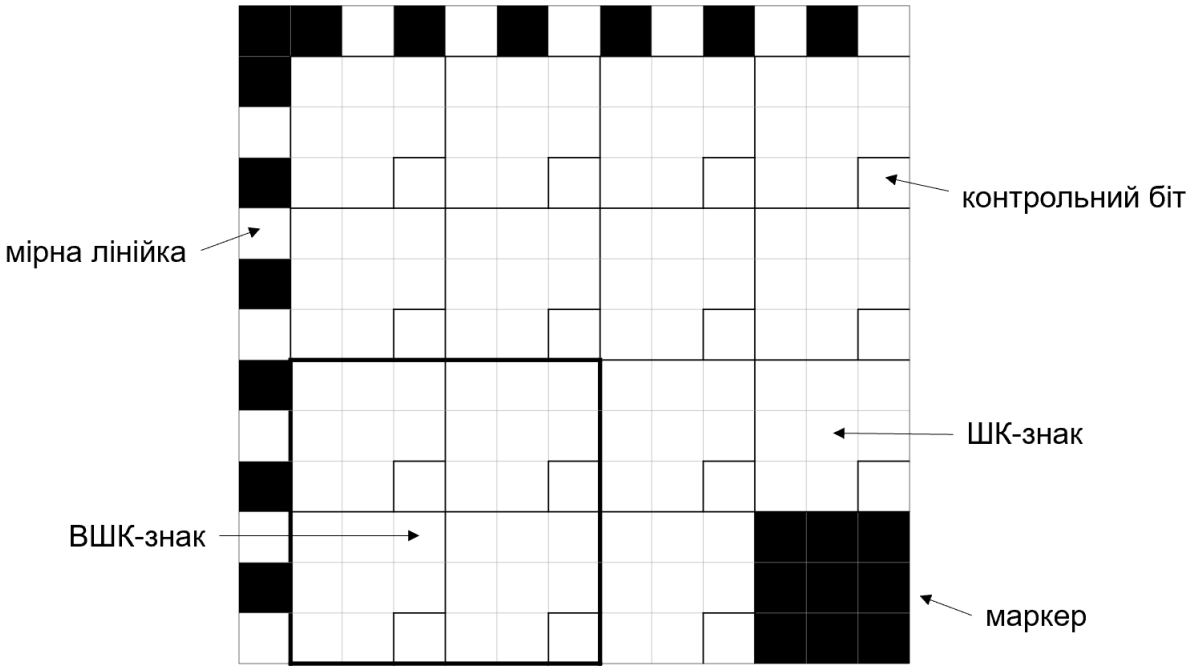


Рис. Г.6. Структура ДДШК-позначки за $s_q = 9$

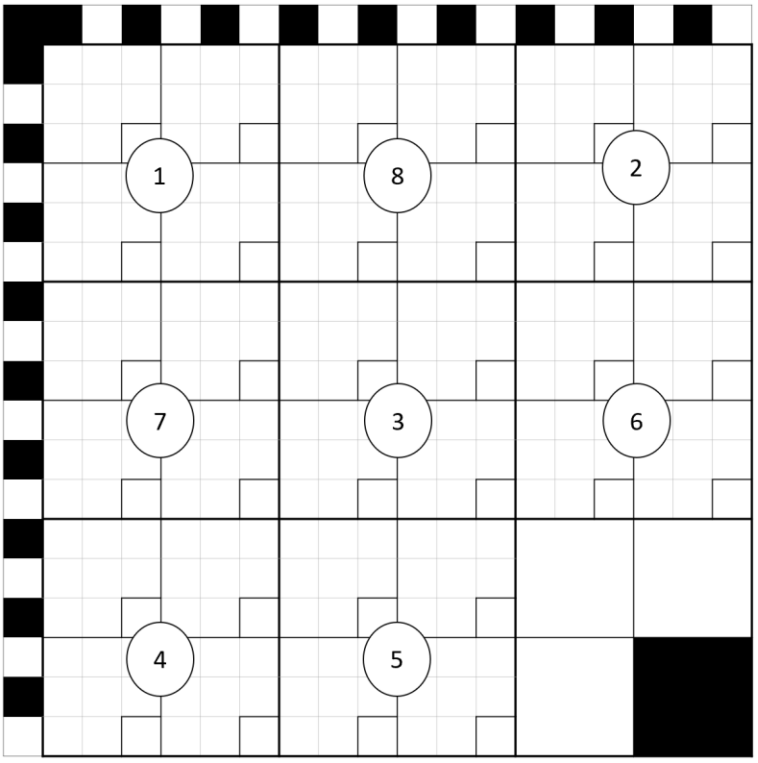
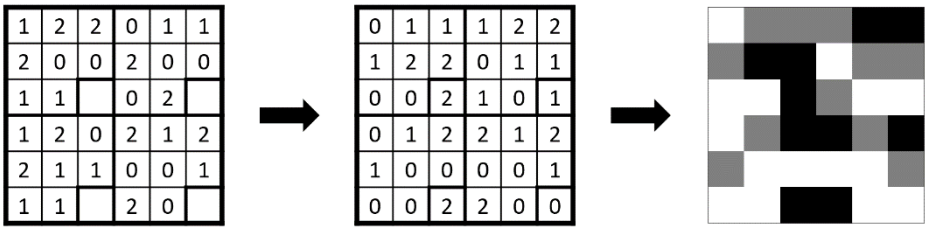


Рис. Г.7. Одна з можливих конфігурацій ДДШК-позначки



(а) (б) (в)

Рис. Г.8. Інвертування комірок у квадранті відповідно до значення контрольного біту: (а) початковий вміст квадрантів, (б) інвертовані значення комірок відповідно до значення контрольних бітів, (в) результуюче представлення трійкової послідовності 0101

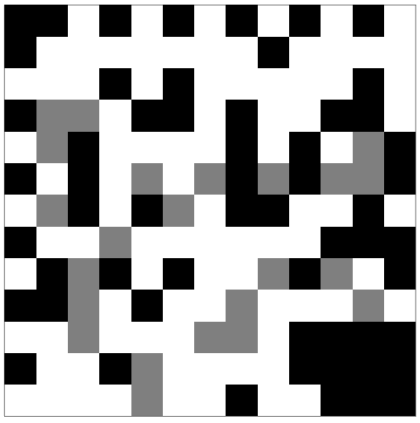


Рис. Г.9. Графічний вигляд нижнього рівня ДШК-позначки

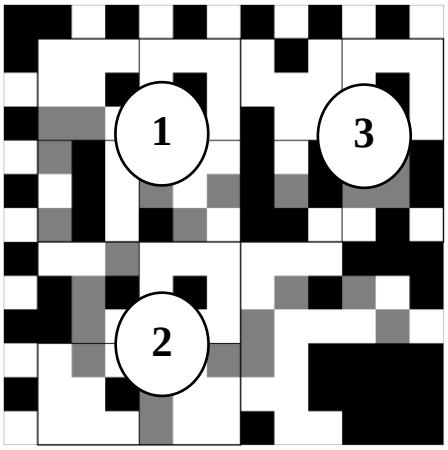


Рис. Г.10. Порядок групування квадрантів у ДДШК-позначці

Рис. Г.11. ДДШК-позначка, що містить закодовану інформацію на нижньому та на верхньому рівнях

Таблиця Г.12. Показники ущільнення для структурного методу та для архіватора WinRAR

Розмір вхідних даних, байти	Показник ущільнення структурного методу	Показник ущільнення архіватора WinRAR
22	1,69	0,24
56	1,70	0,44
118	1,74	0,68
136	1,74	0,72
200	1,77	0,89
300	1,80	1,16
400	1,79	1,17
438	1,80	1,22
767	1,79	1,51
964	1,79	1,59
1389	1,80	1,78
1581	1,80	1,95
2329	1,80	2,12
3573	1,80	2,43
4454	1,80	2,49
6028	1,80	2,70

Додаток Ґ. Програмний код процедури формування множини розв’язків системи нерівностей (2.7) мовою C#

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ObtainingAlphabets
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void culculateButton_Click(object sender, EventArgs e)
        {
            ClearGrid();

            int s = Convert.ToInt32(definingS.Text);
            var arrayToAnalyze = GetAllSolutions(s, 11);

            List<double[]> tableToOutput
                = AnalyzingResults.analyzingResultsInTable(arrayToAnalyze);

            OutputToGrid(tableToOutput);
        }

        private void ClearGrid()
        {
            if (resultingTable.Rows.Count >= 1)
            {
                do
                {
                    try
                    {
                        foreach (DataGridViewRow row in resultingTable.Rows)
                        {
                            resultingTable.Rows.Remove(row);
                        }
                    }
                    catch { }
                } while (resultingTable.Rows.Count > 1);
            }
        }

        private void OutputToGrid(List<double[]> tableToOutput)
        {
            #region OutputDataToDataGrid
            string[] strRow = new string[4];

            foreach (double[] row in tableToOutput)
            {
                strRow[0] = row[0].ToString();
                strRow[1] = row[1].ToString();
                strRow[2] = row[2].ToString();
```

```
        strRow[3] = row[3].ToString();

        resultingTable.Rows.Add(strRow);
    }
    #endregion
}

private double[,] GetAllSolutions(int digitCapacity, int pOmegaAux)
{
    int pOmega = Convert.ToInt32(Math.Pow(3, digitCapacity));

    int pOmegaInf = pOmega - pOmegaAux;
    double log3POmegaInf = Math.Ceiling(Math.Log(pOmegaInf, 3));

    double log3pA, compressionCoefficient;

    List<double[]> resultingTableList = new List<double[]>();

    for (int pA = 10; pA <= 300; pA++)
    {
        for (int n = 1; n <= 20; n++)
        {
            for (int m = 1; m <= 10; m++)
            {
                log3pA = Math.Ceiling(Math.Log(pA, 3));

                if ((Math.Pow(pA, n) <= Math.Pow(pOmegaInf, m)) &&
                    (n * log3pA > m * log3POmegaInf))
                {
                    compressionCoefficient
                        = (n * log3pA) / (m * log3POmegaInf);

                    if ((compressionCoefficient > 1) && (m < n))
                    {
                        resultingTableList.Add(new double[4]
                            { n, m, pA, compressionCoefficient });
                    }
                }
            }
        }
    }

    double[,] arrayToAnalyze = new double[resultingTableList.Count, 4];
    int k = 0;

    foreach (double[] row in resultingTableList)
    {
        arrayToAnalyze[k, 0] = row[0];
        arrayToAnalyze[k, 1] = row[1];
        arrayToAnalyze[k, 2] = row[2];
        arrayToAnalyze[k, 3] = row[3];
        k++;
    }

    return arrayToAnalyze;
}

private void ToCsV(DataGridView dGV, string filename)
{
    string stOutput = "";
    string sHeaders = "";
```

```
    for (int j = 0; j < dGV.Columns.Count; j++)
        sHeaders = sHeaders.ToString() +
            Convert.ToString(dGV.Columns[j].HeaderText) + "\t";
    stOutput += sHeaders + "\r\n";

    for (int i = 0; i < dGV.RowCount - 1; i++)
    {
        string stLine = "";
        for (int j = 0; j < dGV.Rows[i].Cells.Count; j++)
            stLine = stLine.ToString() +
                Convert.ToString(dGV.Rows[i].Cells[j].Value) + "\t";
        stOutput += stLine + "\r\n";
    }
    Encoding utf16 = Encoding.GetEncoding(1254);
    byte[] output = utf16.GetBytes(stOutput);
    FileStream fs = new FileStream(filename, FileMode.Create);
    BinaryWriter bw = new BinaryWriter(fs);
    bw.Write(output, 0, output.Length);
    bw.Flush();
    bw.Close();
    fs.Close();
}

private void saveButton_Click(object sender, EventArgs e)
{
    #region SaveFile
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Excel Documents (*.xls)|*.xls";
    sfd.FileName = "result.xls";
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        ToCsV(resultingTable, sfd.FileName);
    }
    #endregion
}

private List<double[]> GetSolutions(int digitCapacity, int pOmegaAux)
{
    int pOmega = Convert.ToInt32(Math.Pow(3, digitCapacity));

    int pOmegaInf = pOmega - pOmegaAux;
    double log3POmegaInf = Math.Ceiling(Math.Log(pOmegaInf, 3));

    double log3pA, compressionCoefficient;

    List<double[]> resultingTableList = new List<double[]>();

    for (int pA = 10; pA <= 256; pA++)
    {
        for (int n = 1; n <= 20; n++)
        {
            for (int m = 1; m <= 10; m++)
            {
                log3pA = Math.Ceiling(Math.Log(pA, 3));

                if ((Math.Pow(pA, n) <= Math.Pow(pOmegaInf, m)) &&
                            (n * log3pA > m * log3POmegaInf))
                {
                    compressionCoefficient =
                        (n * log3pA) / (m * log3POmegaInf);

                    if ((compressionCoefficient > 1) && (m < n))
```

```
                                resultingTableList.Add(new double[4]
                                        { n, m, pA, compressionCoefficient });
                            }
                        }
                    }
                }

                return resultingTableList;
            }
        }
    }
```

Додаток Д. Програмний код бібліотеки 3GCCodeLibrary мовою C#

```
using System.Collections.Generic;
using BGWCodeLibrary.NotationTranslation;
using BGWCodeLibrary.ReedSolomon.NoiselessDataProcessing;
using BGWCodeLibrary.ReedSolomon.ServingMethods;

namespace BGWCodeLibrary.DataEncoding
{
    /// <summary>
    /// Безпосереднє отримання трійкової послідовності
    /// для представлення у вигляді штрихкодових позначек
    /// </summary>
    public class InputDataToTernaryCodeEncoding
    {
        string AlphabetCodesOfInputData { get; set; }

        DataCodingSettings EncodingSettings { get; set; }
        ComposingGF ComposingGF { get; set; }
        BasesToTranslateBetween Bases { get; set; }

        public InputDataToTernaryCodeEncoding(string alphabetCodesOfInputData,
            DataCodingSettings encodingSettings,
            ComposingGF composingGF, BasesToTranslateBetween bases)
        {
            AlphabetCodesOfInputData = alphabetCodesOfInputData;
            EncodingSettings = encodingSettings;
            ComposingGF = composingGF;
            Bases = bases;
        }

        /// <summary>
        /// Перетворення початкових вхідних даних на трійкову послідовність
        /// </summary>
        public TernaryCode EncodingFromDataToTernaryCode
            (ref LinkedList<int> dataEncodedWithReedSolomon)
        {
            NotationTranslation.NotationTranslation notationTranslation =
                new NotationTranslation.NotationTranslation
                (Bases.InitialBase, Bases.DesirableBase, AlphabetCodesOfInputData);

            LinkedList<int> translatedNumber
                = notationTranslation.TranslationFromNToM(AlphabetCodesOfInputData);

            /* Reed-Solomon encoding */
            ReedSolomonCoding reedSolomonCoding
                = new ReedSolomonCoding(translatedNumber, EncodingSettings,
ComposingGF);
            dataEncodedWithReedSolomon =
reedSolomonCoding.GetReedSolomonEncodedSequence();
            var ternaryCodeAsString
                =
notationTranslation.TranslationToTernaryCode(dataEncodedWithReedSolomon);

            TernaryCode ternaryCode = new TernaryCode(ComposingGF.DigitCapacity);
            ternaryCode.GetSymbol(ternaryCodeAsString);

            return ternaryCode;
        }
    }
```

```
}


using System.Linq;
using BGWCodeLibrary.Exceptions;
using BGWCodeLibrary.ServingMethods;

namespace BGWCodeLibrary.AlphabetProcessing
{
    /// <summary>
    /// Перетворення вхідних даних на інформаційно-кодову послідовність,
    /// де кожний елемент - код символу з відповідного алфавіту
    /// </summary>
    public class AlphabetProcessing
    {
        readonly Alphabet alphabet;
        readonly string actualData;

        public AlphabetProcessing(Alphabet alphabet, string actualData)
        {
            this.alphabet = alphabet;
            this.actualData = actualData;
        }

        public string TransformingActualDataToAlphabetCodes()
        {
            var elements = alphabet.Elements.ToList();
            string alphabetCodesOfActualData = "";

            bool existsInAlphabet = false;

            int i = 0;
            while (i < actualData.Length)
            {
                if (elements.Contains(actualData[i].ToString()))
                {
                    if (elements.IndexOf(actualData[i].ToString()) > 9)
                    {
                        alphabetCodesOfActualData +=
                            "‘" + elements.IndexOf(actualData[i].ToString()) + "‘";
                    }
                    else
                    {
                        alphabetCodesOfActualData
                            += elements.IndexOf(actualData[i].ToString());
                    }
                    existsInAlphabet = true;
                }

                if (!existsInAlphabet)
                {
                    throw new ElementDoesNotExistInAlphabetException
                        ("There is no such element in the alphabet");
                }

                existsInAlphabet = false;
                i++;
            }

            return alphabetCodesOfActualData;
        }
    }
```

```
}


using System.Collections.Generic;
using Nito.Collections;

namespace BGWCodeLibrary.DataEncoding
{
    public class TernaryPattern
    {
        /// <summary>
        /// One pattern represented as a ternary sequence
        /// </summary>
        public Deque<int> TernarySequence { get; private set; }

        public TernaryPattern(LinkedList<int> pattern, int digitCapacity)
        {
            TernarySequence = new Deque<int>(digitCapacity);

            foreach (var digit in pattern)
            {
                TernarySequence.AddToBack(digit);
            }
        }
    }

    public class TernaryCode
    {
        /// <summary>
        /// Entire symbol represented as a set of ternary patterns
        /// </summary>
        public List<TernaryPattern> TernarySymbol { get; set; }
            = new List<TernaryPattern>();

        readonly int digitCapacity;

        public TernaryCode(int digitCapacity)
        {
            this.digitCapacity = digitCapacity;
        }

        TernaryPattern GetPattern(string ternarySequence)
        {
            LinkedList<int> ternarySequencePattern = new LinkedList<int>();

            foreach (var ternaryDigit in ternarySequence)
            {
                ternarySequencePattern.AddLast
                    (int.Parse(ternaryDigit.ToString()));
            }

            while (ternarySequencePattern.Count < digitCapacity)
            {
                ternarySequencePattern.AddFirst(0);
            }

            TernaryPattern pattern
                = new TernaryPattern(ternarySequencePattern, digitCapacity);

            return pattern;
        }
```

```
        public void GetSymbol(LinkedList<string> ternaryCodeString)
        {
            foreach (var code in ternaryCodeString)
            {
                TernarySymbol.Add(GetPattern(code));
            }
        }
    }
}


using System;

namespace BGWCodeLibrary.Exceptions
{
    public class ElementDoesNotExistInAlphabetException : Exception
    {
        public ElementDoesNotExistInAlphabetException()
        {
        }

        public ElementDoesNotExistInAlphabetException(string message)
            : base(message)
        {
        }

        public ElementDoesNotExistInAlphabetException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}


using System;

namespace BGWCodeLibrary.Exceptions
{
    class ImpossibleToFixErrorsException : Exception
    {
        public ImpossibleToFixErrorsException() { }

        public ImpossibleToFixErrorsException(string message) : base(message) { }

        public ImpossibleToFixErrorsException(string message, Exception inner) :
base(message, inner) { }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BGWCodeLibrary.Exceptions
{
    public class NoDataToEncodeException : Exception
    {
        public NoDataToEncodeException()
```

```
        {
        }

        public NoDataToEncodeException(string message)
            : base(message)
        {
        }

        public NoDataToEncodeException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}


using System;

namespace BGWCodeLibrary.Exceptions
{
    class WrongSizeOfMatrixException : Exception
    {
        public WrongSizeOfMatrixException() { }
        public WrongSizeOfMatrixException(string message) : base(message) { }
        public WrongSizeOfMatrixException(string message, Exception inner) :
base(message, inner) { }
    }
}


namespace BGWCodeLibrary.NotationTranslation
{
    public class BasesToTranslateBetween
    {
        public int InitialBase { get; set; }
        public int DesirableBase { get; set; }

        public BasesToTranslateBetween(int initialBase, int desirableBase)
        {
            InitialBase = initialBase;
            DesirableBase = desirableBase;
        }

        public BasesToTranslateBetween() { }
    }
}


using System.Collections.Generic;

namespace BGWCodeLibrary.NotationTranslation
{
    public class NotationTranslation
    {
        private readonly int initialBase;
        private readonly int desirableBase;
        private readonly string stringNumberToTranslate;

        public NotationTranslation
            (int initialBase, int desirableBase, string stringNumberToTranslate)
        {
```

```
    this.initialBase = initialBase;
    this.desirableBase = desirableBase;
    this.stringNumberToTranslate = stringNumberToTranslate;
}

public NotationTranslation(int initialBase, int desirableBase)
{
    this.initialBase = initialBase;
    this.desirableBase = desirableBase;
}

/// <summary>
/// Переведення числа з системи числення з основою N
/// у систему числення з основою M
/// ОСНОВНИЙ МЕТОД
/// </summary>
/// <param name="stringNumberToTranslate"></param>
/// <returns></returns>
public LinkedList<int> TranslationFromNToM(string stringNumberToTranslate)
{
    var numberTranslation = new NumberTranslationFromNToM
        (initialBase, desirableBase, stringNumberToTranslate);

    if (initialBase == 10)
    {
        return numberTranslation.TranslationWithBase10(false);
    }
    else if (desirableBase == 10)
    {
        return numberTranslation.TranslationWithBase10(true);
    }
    else
    {
        return numberTranslation.TranslationFromNToM();
    }
}

/// <summary>
/// Отримання трійкового коду, який відповідає початковим даним, що кодуються
/// </summary>
/// <returns></returns>
public LinkedList<string> TranslationToTernaryCode
    (LinkedList<int> numberToTranslate)
{
    var resultingTernaryCode = new LinkedList<string>();

    foreach (var number in numberToTranslate)
    {
        var numberTranslation
            = new NumberTranslationFromNToM(10, 3, number.ToString());
        var ternaryNumber = numberTranslation.TranslationWithBase10(false);

        string stringTernaryNumber = "";
        foreach(int digit in ternaryNumber)
        {
            stringTernaryNumber += digit.ToString();
        }

        resultingTernaryCode.AddLast(stringTernaryNumber);
    }

    return resultingTernaryCode;
```

```
        }
    }
}


using System;
using System.Collections.Generic;
using System.Numerics;
using BGWCodeLibrary.ServingMethods;

namespace BGWCodeLibrary.NotationTranslation
{
    class NumberTranslationFromNToM
    {
        private readonly int n;
        private readonly int m;
        private readonly string numberToTranslate;

        NumbersReformatting numbersReformatting = new NumbersReformatting();

        public NumberTranslationFromNToM(int n, int m, string numberToTranslate)
        {
            this.n = n;
            this.m = m;
            this.numberToTranslate = numberToTranslate;
        }

        /// <summary>
        /// Переведення з системи числення з основою N у десяткову систему числення
        /// Отримуємо в результаті ОДНЕ десяткове число
        /// </summary>
        /// <param name="numberInNotationN"></param>
        /// <returns></returns>
        private BigInteger TranslationFromNTo10(string numberInNotationN)
        {
            var digitsFromNumber
                = numbersReformatting.DecompositionOfMultidigitNumberToDigits
                (numberInNotationN);
            BigInteger numberInNotation10 = 0;

            int i = digitsFromNumber.Count - 1;

            foreach(var number in digitsFromNumber)
            {
                numberInNotation10 += (BigInteger)(number * Math.Pow(n, i));
                i--;
            }

            return numberInNotation10;
        }

        /// <summary>
        /// Переведення з десяткової системи числення у систему числення з основою M
        /// Отримуємо в результаті число, що складається з окремих цифр
        /// у системі числення з основою M
        /// </summary>
        /// <param name="numberInNotation10"></param>
        /// <returns></returns>
        private LinkedList<int> TranslationFrom10ToM(BigInteger numberInNotation10)
        {
            BigInteger initialNumber = numberInNotation10;
            BigInteger intermediateNumber;
```

```
            BigInteger wholeNumber;

            LinkedList<int> finalNumber = new LinkedList<int>();

            while (true)
            {
                wholeNumber = initialNumber / m;
                intermediateNumber = initialNumber % m;

                finalNumber.AddFirst((int)intermediateNumber);

                if (wholeNumber < m)
                {
                    finalNumber.AddFirst((int)wholeNumber);
                    break;
                }
                else
                {
                    initialNumber = wholeNumber;
                }
            }

            return finalNumber;
        }

        /// <summary>
        /// Подвійне перетворення:
        /// З системи числення N у систему числення M через десяткову систему числення
        /// Отримуємо в результаті число, що складається з окремих цифр
        /// у системі числення з основою M
        /// </summary>
        /// <returns></returns>
        public LinkedList<int> TranslationFromNToM()
        {
            return TranslationFrom10ToM(TranslationFromNTo10(numberToTranslate));
        }

        /// <summary>
        /// Перетворення числа у разі, якщо одна зі систем числення - десяткова
        /// </summary>
        /// <param name="isItFromNTo10"></param>
        /// <returns></returns>
        public LinkedList<int> TranslationWithBase10(bool isItFromNTo10)
        {
            LinkedList<int> translatedNumber;

            if (isItFromNTo10)
            {
                translatedNumber
                    = numbersReformatting.DecompositionOfMultidigitNumberToDigits
                    (Convert.ToString(TranslationFromNTo10(numberToTranslate)));
            }
            else
            {
                translatedNumber = TranslationFrom10ToM(int.Parse(numberToTranslate));
            }

            return translatedNumber;
        }
    }
}
```

```
using System.Collections.Generic;
using BGWCodeLibrary.ReedSolomon.PolynomialProcessing;
using BGWCodeLibrary.ReedSolomon.ServingMethods;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.NoiselessDataProcessing
{
    /// <summary>
    /// Завадостійке кодування Ріда-Соломона
    /// </summary>
    class ReedSolomonCoding
    {
        private PolynomialMultiplier polynomialMultiplier;

        // кодослово - тобто вхідні дані у системі числення P_Omega
        private LinkedList<int> inputDataRepresentedAsAlphabetCodes;

        // r - кількість контрольних кодовекторів
        private int checkSymbolsInCodeword;
        private ComposingGF composingGF; // елементи поля Галуа
        private DataCodingSettings encodingSettings;

        public ReedSolomonCoding(LinkedList<int> inputDataRepresentedAsAlphabetCodes,
            DataCodingSettings encodingSettings, ComposingGF composingGF)
        {
            this.encodingSettings = encodingSettings;
            checkSymbolsInCodeword = encodingSettings.NumberOfControlCodewords;
            this.composingGF = composingGF;
            this.inputDataRepresentedAsAlphabetCodes =
inputDataRepresentedAsAlphabetCodes;
            polynomialMultiplier = new PolynomialMultiplier(composingGF);
        }

        /// <summary>
        /// Отримання кодової послідовності Ріда-Соломона з вхідних даних
        /// </summary>
        public LinkedList<int> GetReedSolomonEncodedSequence()
        {
            TranslatingBetweenPolynomialAndCodewords
                translatingBetweenPolynomialAndCodewords =
                new TranslatingBetweenPolynomialAndCodewords(composingGF);
            PolynomialProcessing.PolynomialProcessing polynomialProcessing =
                new PolynomialProcessing.PolynomialProcessing(composingGF);

            var polynomialBrackets = new List<PolynomialBracket>();

            var polynomial = new Polynomial
            {
                NumberOfBrackets = checkSymbolsInCodeword,
                polynomialBrackets = polynomialBrackets
            };

            /* будуємо твірний многочлен g(x) при r = checkSymbolsInCodeword */
            for (int i = 1; i <= checkSymbolsInCodeword; i++)
            {
                var x = new PolynomialTerm
                {
                    Power = 1,
                    Coefficient = 1
                };

                var xTerms = new List<PolynomialTerm> { x };
```

```
                var alpha = new PolynomialTerm
                {
                    Power = i,
                    Coefficient = -1
                };

                var alphaTerms = new List<PolynomialTerm> { alpha };

                var bracket = new PolynomialBracket
                {
                    xTerms = xTerms,
                    xAlphaTerms = null,
                    alphaTerms = alphaTerms
                };

                polynomial.polynomialBrackets.Add(bracket);
            }

            var firstBracket = polynomial.polynomialBrackets[0];

            /* обчислюємо твірний многочлен */
            for (int i = 1; i < polynomial.NumberOfBrackets; i++)
            {
                var secondBracket = polynomial.polynomialBrackets[i];
                firstBracket
                    = polynomialMultiplier.MultiplyingTwoBrackets
                    (firstBracket, secondBracket);
            }

            var generatorPolynomial
                =
polynomialProcessing.PolynomialAdditionInCommonMiltiplierToGetAlphaPolynomial
                (firstBracket); // твірний многочлен g(x)
            encodingSettings.GeneratorPolynomial = generatorPolynomial;

            var informationalPolynomial
                =
translatingBetweenPolynomialAndCodewords.ObtainPolynomialFromCodeword
                (inputDataRepresentedAsAlphabetCodes);

            /* обчислення кодового многочлену c(x) = a(x)g(x),
             * де a(x) - інформаційний многочлен, тобто вхідна послідовність даних*/
            var codePolynomial
                =
polynomialProcessing.PolynomialAdditionInCommonMiltiplierToGetAlphaPolynomial
                (polynomialMultiplier.MultiplyingTwoBrackets
                (informationalPolynomial, generatorPolynomial));

            /* результуюча кодова послідовність закодованих вхідних даних */
            return
translatingBetweenPolynomialAndCodewords.ObtainCodewordFromPolynomial
                (codePolynomial);
        }
    }
}

using System.Collections.Generic;
using System.Linq;
using BGWCodeLibrary.Exceptions;
using BGWCodeLibrary.NotationTranslation;
using BGWCodeLibrary.ReedSolomon.PolynomialProcessing;
```

```
using BGWCodeLibrary.ReedSolomon.ServingMethods;
using BGWCodeLibrary.ServingMethods;
using BGWCodeLibrary.ServingMethods.Maths;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.NoiselessDataProcessing
{
    public class ReedSolomonDecoding
    {
        // data in P_Omega notation (e.g. 729 for s=6)
        private LinkedList<int> encodedDataInPOmegaNotation;
        public LinkedList<int> CorrectedDataToBeDecoded { get; set; }

        BasesToTranslateBetween Bases { get; }
        DataCodingSettings encodingSettings;
        ComposingGF composingGF;
        PolynomialProcessing.PolynomialProcessing polynomialProcessing;
        PolynomialMultiplier polynomialMultiplier;
        PolynomialDivisor polynomialDivisor;
        AlphaPowerCalculation alphaPowerCalculation;
        TranslatingBetweenPolynomialAndCodewords
translatingBetweenPolynomialAndCodewords;
        NotationTranslation.NotationTranslation notationTranslation;

        private List<int> erasureLocators = new List<int>();
        private int numberOfErasures; // δ

        // χ - максимально можлива кількість помилок для виправлення
        private int maxNumberOfErrors;
        private int numberOfErrors; // γ - фактична кількість помилок

        public ReedSolomonDecoding(LinkedList<int> encodedDataInPOmegaNotation,
            DataCodingSettings encodingSettings, ComposingGF composingGF,
            BasesToTranslateBetween bases)
        {
            this.encodedDataInPOmegaNotation = encodedDataInPOmegaNotation;
            this.encodingSettings = encodingSettings;
            this.composingGF = composingGF;
            Bases = bases;
            numberOfErasures = 0;
            numberOfErrors = 0;
            maxNumberOfErrors = 0;

            polynomialProcessing
                = new PolynomialProcessing.PolynomialProcessing(composingGF);
            polynomialMultiplier = new PolynomialMultiplier(composingGF);
            alphaPowerCalculation = new AlphaPowerCalculation(composingGF);
            translatingBetweenPolynomialAndCodewords
                = new TranslatingBetweenPolynomialAndCodewords(composingGF);
            polynomialDivisor = new PolynomialDivisor(composingGF);
            notationTranslation
                = new NotationTranslation.NotationTranslation
                (Bases.DesirableBase, Bases.InitialBase);
        }

        /// <summary>
        /// Get original data from Reed-Solomon encoded sequence
        /// </summary>
        /// <returns></returns>
        public LinkedList<int> GetInitialDataFromEncodedSequence()
        {
            // визначаємо кількість стирань та розряди, в яких ці стирання мають місце
```

```
            CheckErasures();

            if (numberOfErasures <= encodingSettings.NumberOfControlCodewords)
            {
                if (numberOfErasures > 0)
                {
                    var erasureLocatorsPolynomial = ObtainErasureLocatorsPolynomial();

                    var syndromePolynomial = ObtainSyndromePolynomial();
                    var modifiedSyndromePolynomial = CalculatePolynomialProductModX
                        (syndromePolynomial, erasureLocatorsPolynomial);

                    var componentsOfErrorSyndrome = ObtainComponentsOfErrorSyndrome
                        (modifiedSyndromePolynomial);

                    var errorLocatorsPolynomial = ObtainErrorLocatorsPolynomial
                        (componentsOfErrorSyndrome);
                    var errorLocators = ObtainErrorLocators
                        (errorLocatorsPolynomial);

                    var distortionValuesPolynomial = CalculatePolynomialProductModX
                        (modifiedSyndromePolynomial, errorLocatorsPolynomial);

                    var distortionLocators = errorLocators;
                    distortionLocators.AddRange(erasureLocators);

                    var distortionValues = ObtainDistortionValues
                        (distortionValuesPolynomial, distortionLocators);

                    CorrectDistortion(distortionValues, distortionLocators);
                }
                else
                {
                    CorrectedDataToBeDecoded = encodedDataInPOmegaNotation;
                }

                var decodedDataPolynomial
                    =
translatingBetweenPolynomialAndCodewords.ObtainPolynomialFromCodeword
                    (CorrectedDataToBeDecoded);
                var generatorPolynomial = encodingSettings.GeneratorPolynomial;

                var decodedCodeSequence
                    = polynomialDivisor.PolynomDivider
                    (decodedDataPolynomial, generatorPolynomial);
                var decodedWord
                    =
translatingBetweenPolynomialAndCodewords.ObtainCodewordFromPolynomial
                    (decodedCodeSequence);

                NumbersReformatting reformatting = new NumbersReformatting();
                var initialData = notationTranslation.TranslationFromNToM

(reformatting.ComposingStringFromListOfMultidigitalNumber(decodedWord));

                return initialData;
            }
            else
            {
                throw new ImpossibleToFixErrorsException
                    ("It is impossible to fix this number of erasures under these
settings." +
```

```
            "Please change the settings first.");
    }
}

/// <summary>
/// Визначення розрядів закодованих даних, в яких містяться стирання
/// </summary>
private void CheckErasures()
{
    int currentDigitPlace = encodedDataInPOmegaNotation.Count - 1;
    foreach (var number in encodedDataInPOmegaNotation)
    {
        if (number == -1)
        {
            erasureLocators.Add(currentDigitPlace);
            numberOfErasures++;
        }
        currentDigitPlace--;
    }
}

/// <summary>
/// Обчислення многочлену локаторів стирань λ(x)
/// </summary>
/// <returns></returns>
private PolynomialBracket ObtainErasureLocatorsPolynomial()
{
    // многочлен локаторів стирань lambda(x)
    var lambdaPolynomial = new Polynomial()
    {
        NumberOfBrackets = numberOfErasures,
        polynomialBrackets = new List<PolynomialBracket>()
    };

    /* формування lambda(x) */
    for (int i = 0; i < lambdaPolynomial.NumberOfBrackets; i++)
    {
        var constantTerm = new PolynomialTerm()
        {
            Power = 0,
            Coefficient = 1
        };

        var xTerm = new PolynomialMixedTerm()
        {
            Power = 1,
            TermCoefficient = new PolynomialTerm()
            {
                Power = erasureLocators[i],
                Coefficient = -1
            }
        };

        var polynomial = new PolynomialBracket()
        {
            alphaTerms = new List<PolynomialTerm> { constantTerm },
            xAlphaTerms = new List<PolynomialMixedTerm> { xTerm }
        };

        lambdaPolynomial.polynomialBrackets.Add(polynomial);
    }
```

```
            foreach (var term in lambdaPolynomial.polynomialBrackets)
            {
                var alpha = new List<PolynomialTerm>
                    { term.xAlphaTerms.First().TermCoefficient };
                term.xAlphaTerms.First().TermCoefficient
                    = polynomialProcessing.TransformAlphaTerms(alpha).First();
            }

            var erasureLocatorsPolynomial = new PolynomialBracket();
            if (lambdaPolynomial.NumberOfBrackets > 1)
            {
                erasureLocatorsPolynomial = lambdaPolynomial.polynomialBrackets[0];
                for (int i = 1; i < lambdaPolynomial.NumberOfBrackets; i++)
                {
                    var secondBracket = lambdaPolynomial.polynomialBrackets[i];
                    erasureLocatorsPolynomial
                        = polynomialMultiplier.MultiplyingTwoBrackets
                        (erasureLocatorsPolynomial, secondBracket);
                }
            }
            else
            {
                erasureLocatorsPolynomial =
lambdaPolynomial.polynomialBrackets.First();
            }

            return erasureLocatorsPolynomial;
        }

        /// <summary>
        /// Визначення кількості помилок χ, які можна виправити під час декодування
        /// </summary>
        private void CheckErrors()
        {
            maxNumberOfErrors = (encodingSettings.NumberOfControlCodewords -
numberOfErasures) / 2;
        }

        /// <summary>
        /// Обчислення синдромного многочлену S(x)
        /// </summary>
        private PolynomialBracket ObtainSyndromePolynomial()
        {
            TranslatingBetweenPolynomialAndCodewords
                translatingBetweenPolynomialAndCodewords
                = new TranslatingBetweenPolynomialAndCodewords(composingGF);
            var codePolynomial
                =
translatingBetweenPolynomialAndCodewords.ObtainPolynomialFromCodeword
                (encodedDataInPOmegaNotation);

            var syndromPolynomial = new PolynomialBracket()
            {
                alphaTerms = new List<PolynomialTerm>(),
                xAlphaTerms = new List<PolynomialMixedTerm>(),
                xTerms = null
            };

            var constantTerm = new PolynomialTerm() { Power = 0, Coefficient = 1 };
            syndromPolynomial.alphaTerms.Add(constantTerm);

            for (int i = 1; i <= encodingSettings.NumberOfControlCodewords; i++)
```

```
            {
                var alphaMultiplier = new PolynomialTerm()
                {
                    Power = i,
                    Coefficient = 1
                };

                var distortionSyndrome
                    = polynomialProcessing.ObtainPolynomialOfAlpha
                    (codePolynomial, alphaMultiplier);
                var ternaryCodeOfDistortionSyndrom
                    = polynomialProcessing.ConvertingAlphaPolynomialToTernaryCode
                    (distortionSyndrome, composingGF.DigitCapacity);

                var xAlpha = new PolynomialMixedTerm()
                {
                    Power = i,
                    TermCoefficient = new PolynomialTerm()
                    {
                        Coefficient = 1,
                        Power
                            =
composingGF.GetGFElement(ternaryCodeOfDistortionSyndrom).AlphaPower
                    }
                };

                syndromPolynomial.xAlphaTerms.Add(xAlpha);
            }

            return syndromPolynomial;
        }

        /// <summary>
        /// Обчислення модифікованого синдромного многочлену спотворень
        /// V(x) = S(x)λ(x) mod x^(r+1)
        /// або многочлену значень спотворень Q(x) = V(x)σ(x) mod x^(r+1)
        /// </summary>
        /// <param name="firstBracket"></param>
        /// <param name="secondBracket"></param>
        private PolynomialBracket CalculatePolynomialProductModX
            (PolynomialBracket firstBracket, PolynomialBracket secondBracket)
        {
            var polynomial
                = polynomialMultiplier.MultiplyingTwoBrackets
                (firstBracket, secondBracket);
            polynomial
                =
polynomialProcessing.PolynomialAdditionInCommonMiltiplierToGetAlphaPolynomial
                (polynomial);

            bool oneMoreLoop = true;
            while (oneMoreLoop)
            {
                oneMoreLoop = false;
                for (int i = 0; i < polynomial.xAlphaTerms.Count; i++)
                {
                    if (polynomial.xAlphaTerms[i].Power >=
(encodingSettings.NumberOfControlCodewords + 1) ||
                        polynomial.xAlphaTerms[i].TermCoefficient.Power ==
int.MinValue)
                    {
                        polynomial.xAlphaTerms.RemoveAt(i);
```

```
                    oneMoreLoop = true;
                }
            }
        }

        return polynomial;
    }

    /// <summary>
    /// Обчислення компонентів D синдрому помилок
    /// </summary>
    /// <param name="modifiedSyndromePolynomial"></param>
    /// <returns></returns>
    private List<PolynomialTerm> ObtainComponentsOfErrorSyndrome
        (PolynomialBracket modifiedSyndromePolynomial)
    {
        CheckErrors();
        int epsilon = 2 * maxNumberOfErrors;

        List<PolynomialTerm> errorSyndromeComponent = new List<PolynomialTerm>();

        for (int i = 1; i <= epsilon; i++)
        {
            var component = GetAlphaByXPower
                (modifiedSyndromePolynomial,
                encodingSettings.NumberOfControlCodewords - 2 * maxNumberOfErrors
+ i);

            errorSyndromeComponent.Add(component);
        }

        return errorSyndromeComponent;
    }

    /// <summary>
    /// Отримання альфа-доданку за степінню x
    /// </summary>
    /// <param name="polynomial"></param>
    /// <param name="xPower"></param>
    /// <returns></returns>
    private PolynomialTerm GetAlphaByXPower(PolynomialBracket polynomial, int
xPower)
    {
        foreach (var term in polynomial.xAlphaTerms)
        {
            if (term.Power == xPower) return term.TermCoefficient;
        }

        return null;
    }

    /// <summary>
    /// Отримання многочлену локаторів помилок σ(x)
    /// </summary>
    /// <param name="componentsOfErrorSyndrome"></param>
    private PolynomialBracket ObtainErrorLocatorsPolynomial(List<PolynomialTerm>
componentsOfErrorSyndrome)
    {
        numberOfErrors = maxNumberOfErrors;
        MatrixDeterminant matrixDeterminant = new MatrixDeterminant(composingGF);
        ServingMatrixMethods matrixMethods = new
ServingMatrixMethods(composingGF);
```

```
            MatrixMultiplication matrixMultiplication = new
MatrixMultiplication(composingGF);

            int[,] matrix;

            while (true)
            {
                matrix = new int[numberOfErrors, numberOfErrors];
                for (int i = 0, k = 0; i < matrix.GetLength(0); i++, k++)
                {
                    for (int j = 0, q = k; j < matrix.GetLength(1); j++, q++)
                    {
                        matrix[i, j] = componentsOfErrorSyndrome[q].Power;
                    }
                }

                matrixDeterminant.GetMatrixDeterminant(matrix);

                if (matrixDeterminant.Determinant >= 0) break;
                numberOfErrors--;
            }

            var inversedMatrix = matrixMethods.ObtainInverseMatrix(matrix);
            var errorSyndromComponentsVector =
GetComponentsOfErrorSyndromVector(componentsOfErrorSyndrome);

            var errorLocators =
matrixMultiplication.MultiplyMatrixByVector(inversedMatrix,
errorSyndromComponentsVector);

            /* формуємо многочлен локаторів помилок */
            var errorLocatorsPolynomial = new PolynomialBracket()
            {
                alphaTerms = new List<PolynomialTerm> { new PolynomialTerm
{ Coefficient = 1, Power = 0 } },
                xAlphaTerms = new List<PolynomialMixedTerm>(),
                xTerms = null
            };

            int count = 1;
            foreach (var item in errorLocators)
            {
                var term = new PolynomialMixedTerm
                {
                    Power = count,
                    TermCoefficient = new PolynomialTerm { Coefficient = 1, Power =
item }
                };
                errorLocatorsPolynomial.xAlphaTerms.Add(term);
            }

            return errorLocatorsPolynomial;
        }

        /// <summary>
        /// Формування вектору з компонентів синдрому помилок
        /// </summary>
        /// <param name="componentsOfErrorSyndrome"></param>
        /// <returns></returns>
        private int[] GetComponentsOfErrorSyndromVector(List<PolynomialTerm>
componentsOfErrorSyndrome)
        {
```

```
            var vector = new List<int>();
            AlphaPowerCalculation alphaPowerCalculation = new
AlphaPowerCalculation(composingGF);

            for (int gamma = numberOfErrors; gamma < componentsOfErrorSyndrome.Count;
gamma++)
            {

vector.Add(alphaPowerCalculation.MultiplyAlphaTermWithCoefficient(componentsOfErrorSyn
drome[gamma].Power, 2));
            }

            return vector.ToArray();
        }

        /// <summary>
        /// Визначення розрядів, в яких містяться помилки
        /// </summary>
        /// <param name="errorLocatorsPolynomial"></param>
        /// <returns></returns>
        private List<int> ObtainErrorLocators(PolynomialBracket
errorLocatorsPolynomial)
        {
            var errorLocators = new List<int>();
            SolveEquation solveEquation = new SolveEquation();

            if (errorLocatorsPolynomial.xAlphaTerms.Count == 1)
            {
                var solution = solveEquation.LinearEquation(errorLocatorsPolynomial);
                errorLocators.Add(solution * (-1));
            }

            return errorLocators;
        }

        /// <summary>
        /// Отримання величин спотворень
        /// </summary>
        /// <param name="distortionValuesPolynomial"></param>
        /// <param name="distortionLocators"></param>
        /// <returns></returns>
        private List<int> ObtainDistortionValues(PolynomialBracket
distortionValuesPolynomial, List<int> distortionLocators)
        {
            var distortionValues = new List<int>();

            for (int beta = 0; beta < distortionLocators.Count; beta++)
            {
                /* отримання чисельника */
                var alphaTerm = new PolynomialTerm
                {
                    Coefficient = 1,
                    Power = distortionLocators[beta] * (-1)
                };
                var q =
polynomialProcessing.ObtainPolynomialOfAlpha(distortionValuesPolynomial, alphaTerm);

                var numenator = polynomialProcessing.SumUpAlphaTerms(q);

                /* отримання знаменника */
                var product = new Polynomial { NumberOfBrackets = 0,
polynomialBrackets = new List<PolynomialBracket>() };
```

```
                for (int j = 0; j < distortionLocators.Count; j++)
                {
                    if (beta != j)
                    {
                        var bracket = new PolynomialBracket
                        {
                            xTerms = null,
                            xAlphaTerms = null,
                            alphaTerms = new List<PolynomialTerm> { new PolynomialTerm
{ Coefficient = 1, Power = 0 } }
                        };

                        var term1 = distortionLocators[beta] * (-1);
                        var term2 = distortionLocators[j];
                        var term = new PolynomialTerm
                        {
                            Coefficient = 1,
                            Power =
alphaPowerCalculation.MultiplyAlphaTermWithCoefficient
                                    (alphaPowerCalculation.MultiplyAlphaTerms(term1,
term2), 2)
                        };
                        bracket.alphaTerms.Add(term);

                        product.polynomialBrackets.Add(bracket);
                        product.NumberOfBrackets++;
                    }
                }

                var firstBracket = product.polynomialBrackets[0];
                if (product.NumberOfBrackets > 1)
                {
                    for (int i = 1; i < product.NumberOfBrackets; i++)
                    {
                        var secondBracket = product.polynomialBrackets[i];
                        firstBracket =
polynomialMultiplier.MultiplyingTwoBrackets(firstBracket, secondBracket);
                    }
                }

                var denominator =
polynomialProcessing.SumUpAlphaTerms(firstBracket.alphaTerms);


distortionValues.Add(alphaPowerCalculation.DivideAlphaTerms(numenator.Power,
denominator.Power));
            }

            return distortionValues;
        }

        /// <summary>
        /// Виправлення спотворень, наявних у закодованих даних
        /// </summary>
        /// <param name="distortionValues"></param>
        /// <param name="distortionLocators"></param>
        private void CorrectDistortion(List<int> distortionValues, List<int>
distortionLocators)
        {
            CorrectedDataToBeDecoded = new LinkedList<int>();
            int count = encodedDataInPOmegaNotation.Count - 1;
            foreach (var code in encodedDataInPOmegaNotation)
```

```
            {
                if (distortionLocators.Contains(count))
                {
                    int indexOfDistortion = distortionLocators.IndexOf(count);
                    if (code == -1)
                    {
                        CorrectedDataToBeDecoded.AddLast(composingGF.GetGFElement
                            (alphaPowerCalculation.MultiplyAlphaTermWithCoefficient
                            (distortionValues[indexOfDistortion], 2),
"alphaPower").DecimalRepresentation);
                    }
                    else
                    {
                        CorrectedDataToBeDecoded.AddLast(composingGF.GetGFElement

(alphaPowerCalculation.SubstractAlphaTerms(composingGF.GetGFElement(code,
"decimal").AlphaPower,
                            distortionValues[indexOfDistortion]),
"alphaPower").DecimalRepresentation);
                    }
                }
                else CorrectedDataToBeDecoded.AddLast(code);
                count--;
            }
        }
    }
}

using System.Collections.Generic;
using BGWCodeLibrary.ReedSolomon.ServingMethods;
using BGWCodeLibrary.ServingMethods;
using BGWCodeLibrary.ServingMethods.Maths;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.PolynomialProcessing
{
    class PolynomialDivisor
    {
        TernaryMaths ternaryOperations = new TernaryMaths();
        PolynomialProcessing polynomialProcessing = new PolynomialProcessing();

        ComposingGF composingGF = new ComposingGF();

        public PolynomialDivisor() { }

        public PolynomialDivisor(ComposingGF composingGF)
        {
            this.composingGF = composingGF;
        }

        /// <summary>
        /// Ділення одного поліному на інший
        /// </summary>
        /// <param name="dividend"></param>
        /// <param name="divisor"></param>
        /// <returns></returns>
        public List<PolynomialTerm> PolynomDivider(List<PolynomialTerm> dividend,
List<PolynomialTerm> divisor)
        {
            if (dividend[0].Power == divisor[0].Power)
            {
                return DividentWithSamePower(divisor);
```

```
            }
            else

            if (dividend[0].Power < divisor[0].Power)
            {
                return DividentWithLessPower(dividend, divisor);
            }
            else
            {
                bool ifItIsFirstDivision = true;
                return DividentWithGreaterPower(dividend, divisor,
ifItIsFirstDivision);
            }
        }

        public PolynomialBracket PolynomDivider(PolynomialBracket dividend,
PolynomialBracket divisor)
        {
            if (dividend.xAlphaTerms[0].Power > divisor.xAlphaTerms[0].Power)
            {
                return DividentWithGreaterPower(dividend, divisor);
            }
            return null;
        }

        private List<PolynomialTerm> DividentWithSamePower(List<PolynomialTerm>
polynomial)
        {
            var remainder = new List<PolynomialTerm>();
            var intermediateRemainder = new List<PolynomialTerm>();

            int coefficient = polynomial[0].Coefficient; // коефіцієнт результату
ділення

            foreach (var a in polynomial)
            {
                var term = new PolynomialTerm(); // доданок

                if (a.Power == polynomial[0].Power) // скорочуємо старші члени
                {
                    continue;
                }
                else
                {
                    term.Power = a.Power;
                    term.Coefficient =
ternaryOperations.TernaryMultiplication(coefficient, a.Coefficient);
                    intermediateRemainder.Add(term);
                }
            }

            foreach (var a in intermediateRemainder)
            {
                // доданок
                var term = new PolynomialTerm
                {
                    Power = a.Power,
                    Coefficient = ternaryOperations.TernarySubstraction(0,
a.Coefficient)
                };

                remainder.Add(term);
```

```
            }

            return remainder;
        }

        private List<PolynomialTerm> DividentWithLessPower(List<PolynomialTerm> alpha,
List<PolynomialTerm> polynomial)
        {
            var remainder = new List<PolynomialTerm>();

            int maxPowerInRemainder = polynomial[1].Power; // максимальний степінь в
остачі
            int alphaPower = alpha[0].Power; // степінь діленого

            while (maxPowerInRemainder >= 0)
            {
                // доданок
                var term = new PolynomialTerm
                {
                    Power = maxPowerInRemainder
                };

                if (maxPowerInRemainder == alphaPower)
                {
                    term.Coefficient = 1; // єдиний член остачі
                }
                else
                {
                    term.Coefficient = 0; // решта - нулі
                }

                remainder.Add(term);
                maxPowerInRemainder--;
            }

            return remainder;
        }

        private List<PolynomialTerm> DividentWithGreaterPower(List<PolynomialTerm>
alpha, List<PolynomialTerm> polynomial, bool ifItIsFirstDivision)
        {
            var remainder = new List<PolynomialTerm>();
            var intermediateRemainder = new List<PolynomialTerm>();

            /* пересвідчуємось, що ділене містить всі можливі степені */

            int coefficient = alpha[0].Coefficient; // коефіцієнт, на який множимо
доданок у дільнику
            int multiplier = alpha[0].Power - polynomial[0].Power; // множник, на який
множитемемо дільник

            foreach (var a in polynomial)
            {
                var term = new PolynomialTerm(); // доданок

                if (a.Power == polynomial[0].Power) // скорочуємо найстарші члени
                {
                    continue;
                }
                else
                {
```

```
                    term.Power = a.Power + multiplier; // множимо множник на член
поліному
                    term.Coefficient =
ternaryOperations.TernaryMultiplication(coefficient, a.Coefficient); // множимо
коефіцієнти

                    intermediateRemainder.Add(term);
                }
            }

            int alphaTermIndex = 1;
            foreach (var a in intermediateRemainder)
            {
                // доданок
                var term = new PolynomialTerm
                {
                    Power = a.Power
                };

                if (ifItIsFirstDivision) // якщо це перше ділення
                {
                    term.Coefficient = ternaryOperations.TernarySubstraction(0,
a.Coefficient);
                }
                else
                {
                    if (a.Power == alpha[alphaTermIndex].Power)
                    {
                        term.Coefficient =
ternaryOperations.TernarySubstraction(alpha[alphaTermIndex].Coefficient,
a.Coefficient);
                        if (alphaTermIndex + 1 < alpha.Count)
                        {
                            alphaTermIndex++;
                        }
                    }
                    else
                    {
                        term.Coefficient = ternaryOperations.TernarySubstraction(0,
a.Coefficient);
                    }
                }

                remainder.Add(term);
            }

            if (remainder[0].Power >= polynomial[0].Power)
            {
                ifItIsFirstDivision = false;
                return DividentWithGreaterPower(remainder, polynomial,
ifItIsFirstDivision);
            }
            else
            {
                return remainder;
            }
        }

        private PolynomialBracket DividentWithGreaterPower(PolynomialBracket
firstBracket, PolynomialBracket secondBracket)
        {
```

```
            AlphaPowerCalculation alphaPowerCalculation = new
AlphaPowerCalculation(composingGF);

            var firstPolynomial =
polynomialProcessing.XAlphaTermsOnlyInBracket(firstBracket);
            var secondPolynomial =
polynomialProcessing.XAlphaTermsOnlyInBracket(secondBracket);

            var result = new PolynomialBracket()
            {
                xTerms = null,
                xAlphaTerms = new List<PolynomialMixedTerm>(),
                alphaTerms = new List<PolynomialTerm>()
            };

            while (true)
            {
                var intermediateRemainder = new List<PolynomialMixedTerm>();

                var resultingTerm = new PolynomialMixedTerm()
                {
                    Power = firstPolynomial[0].Power - secondPolynomial[0].Power,
                    TermCoefficient = new PolynomialTerm()
                    {
                        Coefficient = 1,
                        Power = firstPolynomial[0].TermCoefficient.Power -
secondPolynomial[0].TermCoefficient.Power
                    }
                };

                if (resultingTerm.Power > 0) result.xAlphaTerms.Add(resultingTerm);
                else
                {
                    var alphaTerm = new PolynomialTerm()
                    {
                        Coefficient = 1,
                        Power = firstPolynomial[0].TermCoefficient.Power -
secondPolynomial[0].TermCoefficient.Power
                    };
                    result.alphaTerms.Add(alphaTerm);
                }

                for (int i = 0, j = 0; j < firstPolynomial.Count; i++, j++)
                {
                    var remainderTerm = new PolynomialMixedTerm();
                    if (j < secondPolynomial.Count)
                    {
                        var intermediateTerm = new PolynomialMixedTerm()
                        {
                            Power = resultingTerm.Power + secondPolynomial[j].Power,
                            TermCoefficient = new PolynomialTerm()
                            {
                                Coefficient = 1,
                                Power = resultingTerm.TermCoefficient.Power +
secondPolynomial[j].TermCoefficient.Power
                            }
                        };

                        if (firstPolynomial[i].TermCoefficient.Power !=
intermediateTerm.TermCoefficient.Power)
                        {
                            remainderTerm.Power = intermediateTerm.Power;
```

```
                            remainderTerm.TermCoefficient = new PolynomialTerm()
                            {
                                Coefficient = 1,
                                Power = alphaPowerCalculation.SubstractAlphaTerms
                                    (firstPolynomial[i].TermCoefficient.Power,
intermediateTerm.TermCoefficient.Power)
                            };
                        }
                        else { continue; }
                    }
                    else
                    {
                        remainderTerm = firstPolynomial[i];
                    }

                    intermediateRemainder.Add(remainderTerm);
                }

                if (intermediateRemainder.Count != 0) firstPolynomial =
intermediateRemainder;
                else break;
            }

            return result;
        }
    }
}

using System;
using System.Collections.Generic;
using BGWCodeLibrary.ReedSolomon.ServingMethods;
using BGWCodeLibrary.ServingMethods;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.PolynomialProcessing
{
    public class PolynomialMultiplier
    {
        TernaryMaths ternaryOperations = new TernaryMaths();
        ComposingGF composingGF = new ComposingGF();

        public PolynomialMultiplier(ComposingGF composingGF)
        {
            this.composingGF = composingGF;
        }

        /// <summary>
        /// Множимо два поліноми
        /// </summary>
        /// <param name="firstBracket"></param>
        /// <param name="secondBracket"></param>
        /// <returns></returns>
        public PolynomialBracket MultiplyingTwoBrackets(PolynomialBracket firstBracket,
PolynomialBracket secondBracket)
        {
            firstBracket = GettingRidOfNegativeTerms(firstBracket);
            secondBracket = GettingRidOfNegativeTerms(secondBracket);

            var xTermsInFirstBracket = firstBracket.xTerms;
            var alphaTermsInFirstBracket = firstBracket.alphaTerms;
            var xAlphaTermsInFirstBracket = firstBracket.xAlphaTerms;
```

```
            var xTermsInSecondBracket = secondBracket.xTerms;
            var alphaTermsInSecondBracket = secondBracket.alphaTerms;
            var xAlphaTermsInSecondBracket = secondBracket.xAlphaTerms;

            var x = new List<PolynomialTerm>();
            var xAlpha = new List<PolynomialMixedTerm>();
            var alpha = new List<PolynomialTerm>();

            foreach (PolynomialTerm xInFirstBracket in xTermsInFirstBracket)
            {
                // множимо x на x-ві компоненти поліному
                if (xTermsInSecondBracket.Count > 0) x =
MultiplyingYByY(xInFirstBracket, xTermsInSecondBracket);
                // x на x-альфа
                if (xAlphaTermsInSecondBracket.Count > 0) xAlpha =
MultiplyingXByXAlpha(xInFirstBracket, xAlphaTermsInSecondBracket);
                // x на альфа
                if (alphaTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingXByAlpha(xInFirstBracket, alphaTermsInSecondBracket));
            }

            foreach (PolynomialTerm alphaInFirstBracket in alphaTermsInFirstBracket)
            {
                // альфа на альфа
                if (alphaTermsInSecondBracket.Count > 0) alpha =
MultiplyingYByY(alphaInFirstBracket, alphaTermsInSecondBracket);
                // альфа на x-альфа
                if (xAlphaTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingAlphaByXAlpha(alphaInFirstBracket,
xAlphaTermsInSecondBracket));
                // альфа на x
                if (xTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingAlphaByX(alphaInFirstBracket, xTermsInSecondBracket));
            }

            foreach (PolynomialMixedTerm xAlphaInFirstBracket in
xAlphaTermsInFirstBracket)
            {
                // x-альфа на x
                if (xTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingXAlphaByX(xAlphaInFirstBracket, xTermsInSecondBracket));
                // x-альфа на x-альфа
                if (xAlphaTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingXAlphaByXAlpha(xAlphaInFirstBracket,
xAlphaTermsInSecondBracket));
                // x-альфа на альфа
                if (alphaTermsInSecondBracket.Count > 0)
xAlpha.AddRange(MultiplyingXAlphaByAlpha(xAlphaInFirstBracket,
alphaTermsInSecondBracket));
            }

            PolynomialBracket resultOfMultiplication = new PolynomialBracket
            {
                xTerms = x,
                xAlphaTerms = xAlpha,
                alphaTerms = alpha
            };

            return resultOfMultiplication;
        }

        /* -------------------- ОПЕРАЦІЇ МНОЖЕННЯ -------------------- */
```

```
        /// <summary>
        /// Множення однорідних елементів
        /// </summary>
        /// <param name="y"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialTerm> MultiplyingYByY(PolynomialTerm y,
List<PolynomialTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialTerm>();

            foreach (var yInBracket in bracket)
            {
                var yy = new PolynomialTerm
                {
                    Power = (y.Power + yInBracket.Power) %
composingGF.NumberOfGFElements,
                    Coefficient =
ternaryOperations.TernaryMultiplication(y.Coefficient, yInBracket.Coefficient)
                };

                resultOfMultiplication.Add(yy);
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення x на x-альфа
        /// </summary>
        /// <param name="x"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingXByXAlpha(PolynomialTerm x,
List<PolynomialMixedTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var xAlphaInBracket in bracket)
            {
                var xAlpha = new PolynomialMixedTerm
                {
                    // степінь x
                    Power = x.Power + xAlphaInBracket.Power,
                    // коефіцієнт при x - тобто альфа
                    TermCoefficient = new PolynomialTerm
                    {
                        // степінь альфа
                        Power = xAlphaInBracket.TermCoefficient.Power,
                        // коефіцієнт альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(x.Coefficient,
xAlphaInBracket.TermCoefficient.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlpha);
            }

            return resultOfMultiplication;
        }
```

```
        /// <summary>
        /// Множення альфа на х-альфа
        /// </summary>
        /// <param name="alpha"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingAlphaByXAlpha(PolynomialTerm
alpha, List<PolynomialMixedTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var xAlphaInBracket in bracket)
            {
                var xAlpha = new PolynomialMixedTerm
                {
                    // степінь х
                    Power = xAlphaInBracket.Power,
                    // коефіцієнт при х - тобто альфа
                    TermCoefficient = new PolynomialTerm
                    {
                        // степінь альфа
                        Power = (alpha.Power +
xAlphaInBracket.TermCoefficient.Power) % composingGF.NumberOfGFElements,
                        // коефіцієнт альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(alpha.Coefficient,
xAlphaInBracket.TermCoefficient.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlpha);
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення х на альфа
        /// </summary>
        /// <param name="x"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingXByAlpha(PolynomialTerm x,
List<PolynomialTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var alphaInBracket in bracket)
            {
                var xAlpha = new PolynomialMixedTerm
                {
                    Power = x.Power, // степінь х
                    TermCoefficient = new PolynomialTerm
                    {
                        Power = alphaInBracket.Power, // степінь альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(x.Coefficient, alphaInBracket.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlpha);
```

```
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення альфа на х
        /// </summary>
        /// <param name="alpha"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingAlphaByX(PolynomialTerm alpha,
List<PolynomialTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var xInBracket in bracket)
            {
                var xAlpha = new PolynomialMixedTerm
                {
                    Power = xInBracket.Power, // степінь х
                    TermCoefficient = new PolynomialTerm
                    {
                        Power = alpha.Power, // степінь альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(alpha.Coefficient, xInBracket.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlpha);
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення х-альфа на х
        /// </summary>
        /// <param name="xAlpha"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingXAlphaByX(PolynomialMixedTerm
xAlpha, List<PolynomialTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var xInBracket in bracket)
            {
                var xAlphaResulting = new PolynomialMixedTerm
                {
                    // степінь х
                    Power = (xInBracket.Power + xAlpha.Power) %
composingGF.NumberOfGFElements,
                    // коефіцієнт при х - тобто альфа
                    TermCoefficient = new PolynomialTerm
                    {
                        // степінь альфа
                        Power = xAlpha.TermCoefficient.Power,
                        // коефіцієнт альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(xAlpha.TermCoefficient.Coefficient,
xInBracket.Coefficient)
```

```
                    }
                };

                resultOfMultiplication.Add(xAlphaResulting);
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення х-альфа на х-альфа
        /// </summary>
        /// <param name="xAlpha"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm>
MultiplyingXAlphaByXAlpha(PolynomialMixedTerm xAlpha, List<PolynomialMixedTerm>
bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var xAlphaInBracket in bracket)
            {
                var xAlphaResulting = new PolynomialMixedTerm
                {
                    // степінь х
                    Power = xAlphaInBracket.Power + xAlpha.Power,
                    // коефіцієнт при х - тобто альфа
                    TermCoefficient = new PolynomialTerm
                    {
                        // степінь альфа
                        Power = (xAlpha.TermCoefficient.Power +
xAlphaInBracket.TermCoefficient.Power) % composingGF.NumberOfGFElements,
                        // коефіцієнт альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(xAlpha.TermCoefficient.Coefficient,
xAlphaInBracket.TermCoefficient.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlphaResulting);
            }

            return resultOfMultiplication;
        }

        /// <summary>
        /// Множення х-альфа на альфа
        /// </summary>
        /// <param name="xAlpha"></param>
        /// <param name="bracket"></param>
        /// <returns></returns>
        private List<PolynomialMixedTerm> MultiplyingXAlphaByAlpha(PolynomialMixedTerm
xAlpha, List<PolynomialTerm> bracket)
        {
            var resultOfMultiplication = new List<PolynomialMixedTerm>();

            foreach (var alphaInBracket in bracket)
            {
                var xAlphaResulting = new PolynomialMixedTerm
                {
                    // степінь х
```

```
                    Power = xAlpha.Power,
                    // коефіцієнт при x - тобто альфа
                    TermCoefficient = new PolynomialTerm
                    {
                        // степінь альфа
                        Power = (xAlpha.TermCoefficient.Power +
alphaInBracket.Power) % composingGF.NumberOfGFElements,
                        // коефіцієнт альфа
                        Coefficient =
ternaryOperations.TernaryMultiplication(xAlpha.TermCoefficient.Coefficient,
alphaInBracket.Coefficient)
                    }
                };

                resultOfMultiplication.Add(xAlphaResulting);
            }

            return resultOfMultiplication;
        }

        /* -------------------- ДОДАТКОВІ ОПЕРАЦІЇ -------------------- */

        /// <summary>
        /// Позбуваємось від від’ємних коефіцієнтів у поліномі
        /// шляхом застосування трійкової арифметики
        /// </summary>
        /// <param name="polynomialWithNegativeTerms"></param>
        /// <returns></returns>
        private PolynomialBracket GettingRidOfNegativeTerms(PolynomialBracket
polynomialWithNegativeTerms)
        {
            var xTermsNegative = polynomialWithNegativeTerms.xTerms;
            var alphaTermsNegative = polynomialWithNegativeTerms.alphaTerms;
            var xAlphaTermsNegative = polynomialWithNegativeTerms.xAlphaTerms;

            var xTermsPositive = new List<PolynomialTerm>();
            var alphaTermsPositive = new List<PolynomialTerm>();
            var xAlphaTermsPositive = new List<PolynomialMixedTerm>();

            foreach (var xTerm in xTermsNegative ?? new List<PolynomialTerm>())
            {
                var xTermPositive = new PolynomialTerm();

                if (xTerm.Coefficient < 0)
                {
                    xTermPositive.Power = xTerm.Power;
                    xTermPositive.Coefficient =
ternaryOperations.TernarySubstraction(0, Math.Abs(xTerm.Coefficient));
                }
                else
                {
                    xTermPositive = xTerm;
                }

                xTermsPositive.Add(xTermPositive);
            }

            foreach (var alphaTerm in alphaTermsNegative ?? new
List<PolynomialTerm>())
            {
                var alphaTermPositive = new PolynomialTerm();
```

```
                if (alphaTerm.Coefficient < 0)
                {
                    alphaTermPositive.Power = alphaTerm.Power;
                    alphaTermPositive.Coefficient =
ternaryOperations.TernarySubstraction(0, Math.Abs(alphaTerm.Coefficient));
                }
                else
                {
                    alphaTermPositive = alphaTerm;
                }

                alphaTermsPositive.Add(alphaTermPositive);
            }

            foreach (var mixedTerm in xAlphaTermsNegative ?? new
List<PolynomialMixedTerm>())
            {
                var mixedTermPositive = new PolynomialMixedTerm();

                if (mixedTerm.TermCoefficient.Coefficient < 0)
                {
                    mixedTermPositive.Power = mixedTerm.Power;
                    mixedTermPositive.TermCoefficient.Coefficient =
ternaryOperations.TernarySubstraction(0,
Math.Abs(mixedTerm.TermCoefficient.Coefficient));
                }
                else
                {
                    mixedTermPositive = mixedTerm;
                }

                xAlphaTermsPositive.Add(mixedTermPositive);
            }

            var polynomialWithPositiveTerms = new PolynomialBracket()
            {
                xTerms = xTermsPositive,
                xAlphaTerms = xAlphaTermsPositive,
                alphaTerms = alphaTermsPositive
            };

            return polynomialWithPositiveTerms;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Headers;
using System.Text;
using BGWCodeLibrary.ReedSolomon.ServingMethods;
using BGWCodeLibrary.ServingMethods;
using BGWCodeLibrary.ServingMethods.Maths;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.PolynomialProcessing
{
    class PolynomialProcessing
    {
        private ComposingGF gfElements = new ComposingGF(); // поле Галуа
```

```
        TranslatingBetweenPolynomialAndCodewords
translatingBetweenPolynomialAndCodewords;
        AlphaPowerCalculation alphaPowerCalculation;

        TernaryMaths operations = new TernaryMaths();

        public PolynomialProcessing(ComposingGF gfElements)
        {
            this.gfElements = gfElements;
            translatingBetweenPolynomialAndCodewords = new
TranslatingBetweenPolynomialAndCodewords(gfElements);
            alphaPowerCalculation = new AlphaPowerCalculation(gfElements);
        }

        public PolynomialProcessing() { }

        /// <summary>
        /// Винесення за дужки спільного множника - x
        /// </summary>
        /// <param name="polynomial"></param>
        private void TakingCommonMultiplierOutOfBrackets(PolynomialBracket polynomial,
out List<CommonMultiplier> commonMultipliers)
        {
            commonMultipliers = new List<CommonMultiplier>();

            var xTerms = polynomial.xTerms;
            var xAlphaTerms = polynomial.xAlphaTerms;

            int maxPower = 0;
            foreach (var term in xTerms ?? new List<PolynomialTerm>())
            {
                if (term.Power > maxPower) maxPower = term.Power;
            }

            foreach (var term in xAlphaTerms ?? new List<PolynomialMixedTerm>())
            {
                if (term.Power > maxPower) maxPower = term.Power;
            }

            while (maxPower > 0)
            {
                var commonMultiplier = new CommonMultiplier();
                commonMultiplier.XPower = maxPower;
                commonMultiplier.commonCoefficient = new List<PolynomialTerm>();

                if (IfPowerExistsInPolynomial(xTerms, maxPower))
                {
                    foreach (var term in xTerms ?? new List<PolynomialTerm>())
                    {
                        if (term.Power == maxPower)
                        {
                            var xCoeff = new PolynomialTerm
                            {
                                Power = 0,
                                Coefficient = term.Coefficient
                            };
                            commonMultiplier.commonCoefficient.Add(xCoeff);
                        }
                    }

                    commonMultipliers.Add(commonMultiplier);
                }
```

```
                if (IfPowerExistsInPolynomial(xAlphaTerms, maxPower))
                {
                    foreach (var term in xAlphaTerms ?? new
List<PolynomialMixedTerm>())
                    {
                        if (term.Power == maxPower)
                        {
                            var xCoeff = new PolynomialTerm
                            {
                                Power = term.TermCoefficient.Power,
                                Coefficient = term.TermCoefficient.Coefficient
                            };
                            commonMultiplier.commonCoefficient.Add(xCoeff);
                        }
                    }

                    commonMultipliers.Add(commonMultiplier);
                }

                maxPower--;
            }
        }

        /// <summary>
        /// Отримання трійкового коду з альфа-поліному
        /// </summary>
        /// <param name="commonMultiplier"></param>
        /// <returns></returns>
        public string ConvertingAlphaPolynomialToTernaryCode(List<PolynomialTerm>
alphasInBracket, int digitCapacity)
        {
            int numberOfAlphas = alphasInBracket.Count;
            string ternarySumOfAlphas = "";

            if (numberOfAlphas == 1)
            {
                var ternaryAlphaRepresentation =
ConvertTernaryStringToLinkedList(gfElements.GetGFElement(alphasInBracket.First().Power,
"alphaPower").TernaryRepresentation);

                foreach (int number in ternaryAlphaRepresentation)
                {
                    ternarySumOfAlphas +=
operations.TernaryMultiplication(alphasInBracket.First().Coefficient,
number).ToString();
                }
            }
            else
            {
                int[,] ternaryNumbersToAdd = new int[numberOfAlphas, digitCapacity];

                int countOfAlphas = 0;
                foreach (var alpha in alphasInBracket)
                {
                    var ternaryAlphaRepresentation =
ConvertTernaryStringToLinkedList(gfElements.GetGFElement(alpha.Power,
"alphaPower").TernaryRepresentation);

                    int countOfDigits = 0;
                    foreach (var number in ternaryAlphaRepresentation)
                    {
```

```
                        ternaryNumbersToAdd[countOfAlphas, countOfDigits] =
operations.TernaryMultiplication(alpha.Coefficient, number);
                        countOfDigits++;
                    }
                    countOfAlphas++;
                }

                for (int i = 1; i < numberOfAlphas; i++) // row
                {
                    for (int j = 0; j < digitCapacity; j++) // columns
                    {
                        ternaryNumbersToAdd[0, j] =
operations.TernaryAddition(ternaryNumbersToAdd[0, j], ternaryNumbersToAdd[i, j]);
                    }
                }

                for (int j = 0; j < digitCapacity; j++)
                {
                    ternarySumOfAlphas += ternaryNumbersToAdd[0, j].ToString();
                }
            }

            return ternarySumOfAlphas;
        }

        /// <summary>
        /// Додавання альфа-доданків, що мають спільний множник x,
        /// для отримання єдиного многочлену від x вигляду (ax + a)
        /// </summary>
        /// <returns></returns>
        public PolynomialBracket
PolynomialAdditionInCommonMiltiplierToGetAlphaPolynomial(PolynomialBracket polynomial)
        {
            List<CommonMultiplier> commonMultipliers = new List<CommonMultiplier>();
            TakingCommonMultiplierOutOfBrackets(polynomial, out commonMultipliers);

            var x = new List<PolynomialTerm>();
            var xAlpha = new List<PolynomialMixedTerm>();
            var alpha = polynomial.alphaTerms;

            foreach (var commonMultiplier in commonMultipliers)
            {
                string ternaryCode =
ConvertingAlphaPolynomialToTernaryCode(commonMultiplier.commonCoefficient,
gfElements.DigitCapacity);

                var term = new PolynomialMixedTerm()
                {
                    Power = commonMultiplier.XPower,
                    TermCoefficient =
translatingBetweenPolynomialAndCodewords.ObtainAlphaTermFromCommonCoefficient(ternaryC
ode)
                };
                xAlpha.Add(term);
            }

            alpha = TransformAlphaTerms(alpha);

            var resultingPolynomial = new PolynomialBracket()
            {
                xTerms = x,
                xAlphaTerms = xAlpha,
```

```
                alphaTerms = alpha
            };

            return resultingPolynomial;
        }

        /// <summary>
        /// Отримання многочлену від альфа (підставляємо альфа замість x)
        /// </summary>
        /// <param name="polynomial"></param>
        /// <returns></returns>
        public List<PolynomialTerm> ObtainPolynomialOfAlpha(PolynomialBracket
polynomial, PolynomialTerm alpha)
        {
            var alphaPolynomial = new List<PolynomialTerm>();

            foreach (var xAlpha in polynomial.xAlphaTerms)
            {
                var alphaTerm = new PolynomialTerm()
                {
                    Power = (xAlpha.Power * alpha.Power +
xAlpha.TermCoefficient.Power) % gfElements.NumberOfGFElements,
                    Coefficient = 1
                };

                alphaPolynomial.Add(alphaTerm);
            }

            alphaPolynomial.AddRange(polynomial.alphaTerms);

            return alphaPolynomial;
        }

        /// <summary>
        /// Сумування однорідних доданків поліному
        /// </summary>
        /// <param name="polynomial"></param>
        /// <returns></returns>
        public PolynomialTerm SumUpAlphaTerms(List<PolynomialTerm> polynomial)
        {
            var firstTerm = polynomial[0];
            for (int k = 1; k < polynomial.Count; k++)
            {
                var secondTerm = polynomial[k];
                firstTerm.Power = alphaPowerCalculation.AddAlphaTerms(firstTerm.Power,
secondTerm.Power);
            }

            return firstTerm;
        }

        /* ---------------- SERVING METHODS ---------------- */

        /// <summary>
        /// Позбуваємось від степеней альфа, роблячи заміну з поля Галуа
        /// </summary>
        public List<PolynomialTerm> TransformAlphaTerms(List<PolynomialTerm> terms)
        {
            var transformedAlphaTerms = new List<PolynomialTerm>();

            foreach (var term in terms)
            {
```

```
                GFElement alpha = gfElements.GetGFElement(term.Power, "alphaPower");

                var transformedTermInPolynomialRepresentation = new
List<PolynomialTerm>();

                int count = gfElements.DigitCapacity - 1;
                foreach (var digit in alpha.TernaryRepresentation)
                {
                    if (digit != '0')
                    {
                        var alphaTerm = new PolynomialTerm()
                        {
                            Coefficient = term.Coefficient < 0 ?

operations.TernaryMultiplication(operations.TernarySubstraction(0,
Math.Abs(term.Coefficient)), Int32.Parse(digit.ToString())) :
                                operations.TernaryMultiplication(term.Coefficient,
Int32.Parse(digit.ToString())),
                            Power = count
                        };

                        transformedTermInPolynomialRepresentation.Add(alphaTerm);
                    }
                    count--;
                }

                var ternaryRepresentationOfAlphaTerm = new
string[gfElements.DigitCapacity];
                foreach (var alphaTerm in transformedTermInPolynomialRepresentation)
                {
                    ternaryRepresentationOfAlphaTerm[gfElements.DigitCapacity -
alphaTerm.Power - 1] = alphaTerm.Coefficient.ToString();
                }
                for (int i = 0; i < ternaryRepresentationOfAlphaTerm.Length; i++)
                {
                    if (ternaryRepresentationOfAlphaTerm[i] == null)
ternaryRepresentationOfAlphaTerm[i] = "0";
                }
                string ternaryCodeOfAlpha = String.Join("",
ternaryRepresentationOfAlphaTerm);

                var transformedAlphaTerm =
translatingBetweenPolynomialAndCodewords.ObtainAlphaTermFromCommonCoefficient(ternaryC
odeOfAlpha);
                transformedAlphaTerms.Add(transformedAlphaTerm);
            }

            return transformedAlphaTerms;
        }

        /// <summary>
        /// Перевірка, чи існує член із заданим степенем в однорідному многочлені
        /// </summary>
        /// <param name="terms"></param>
        /// <param name="power"></param>
        /// <returns></returns>
        private bool IfPowerExistsInPolynomial(List<PolynomialTerm> terms, int power)
        {
            foreach (var term in terms)
            {
                if (term.Power == power) return true;
            }
```

```
            return false;
        }

        /// <summary>
        /// Перевірка, чи існує член із заданим степенем у неоднорідному многочлені
        /// </summary>
        /// <param name="terms"></param>
        /// <param name="power"></param>
        /// <returns></returns>
        private bool IfPowerExistsInPolynomial(List<PolynomialMixedTerm> terms, int
power)
        {
            foreach (var term in terms)
            {
                if (term.Power == power) return true;
            }

            return false;
        }

        /// <summary>
        /// Перетворення рядку трійкового коду на LinkedList типу int
        /// </summary>
        /// <param name="ternaryString"></param>
        /// <returns></returns>
        private LinkedList<int> ConvertTernaryStringToLinkedList(string ternaryString)
        {
            var ternaryLinkedList = new LinkedList<int>();

            foreach (var digit in ternaryString)
            {
                ternaryLinkedList.AddLast(Int32.Parse(digit.ToString()));
            }

            return ternaryLinkedList;
        }

        /// <summary>
        /// Залишаємо у поліномі лише x-альфа члени (перетворюємо альфа-член на x-
альфа член)
        /// </summary>
        /// <param name="bracket"></param>
        /// <returns></returns>
        public List<PolynomialMixedTerm> XAlphaTermsOnlyInBracket(PolynomialBracket
bracket)
        {
            var xAlphaZeroPower = new PolynomialMixedTerm()
            {
                Power = 0,
                TermCoefficient = new PolynomialTerm()
                {
                    Power = bracket.alphaTerms.First().Power,
                    Coefficient = bracket.alphaTerms.First().Coefficient
                }
            };

            bracket.xAlphaTerms.Add(xAlphaZeroPower);

            return bracket.xAlphaTerms;
        }
    }
```

```
}

using System.Collections.Generic;

namespace BGWCodeLibrary.ReedSolomon.PolynomialProcessing
{
    public class PolynomialStructure
    {
        public class PolynomialTerm
        {
            public int Power { get; set; }
            public int Coefficient { get; set; }
        }

        public class PolynomialMixedTerm
        {
            // коефіцієнт для x - альфа
            public PolynomialTerm TermCoefficient { get; set; } = new
PolynomialTerm();

            // степінь для x
            public int Power { get; set; }
        }

        public class PolynomialBracket
        {
            public List<PolynomialTerm> xTerms;
            public List<PolynomialTerm> alphaTerms;
            public List<PolynomialMixedTerm> xAlphaTerms;
        }

        public class Polynomial
        {
            public int NumberOfBrackets { get; set; }
            public List<PolynomialBracket> polynomialBrackets;
        }

        public class GFElement
        {
            public int AlphaPower { get; set; }
            public int NegativeAlphaPower { get; set; }
            public string TernaryRepresentation { get; set; }
            public int DecimalRepresentation { get; set; }
        }

        public class CommonMultiplier
        {
            public int XPower { get; set; }
            public List<PolynomialTerm> commonCoefficient;
        }
    }
}


using System;
using System.Collections.Generic;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;
using BGWCodeLibrary.ServingMethods;
using BGWCodeLibrary.ReedSolomon.PolynomialProcessing;

namespace BGWCodeLibrary.ReedSolomon.ServingMethods
{
```

```
    public class ComposingGF
    {
        public LinkedList<GFElement> GFElements { get; }
        public int DigitCapacity { get; }
        public int NumberOfGFElements { get; }

        /// <summary>
        /// Отримання повного подання елементів поля Галуа за модулем незвідного
многочлена
        /// </summary>
        /// <param name="polynomial"></param>
        /// <param name="NumberOfGFElements"></param>
        /// <returns></returns>
        public ComposingGF(string polynomial, int numberOfElements)
        {
            var notationTranslation = new NotationTranslation.NotationTranslation(3,
10);
            var numbersReformatting = new NumbersReformatting();

            PolynomialDivisor polynomialDivisor = new PolynomialDivisor();
            LinkedList<GFElement> gFElementsList = new LinkedList<GFElement>();

            var divisor = DecodePolynomialNotation(polynomial);

            int i = 0;
            NumberOfGFElements = numberOfElements - 1;

            while (i < NumberOfGFElements)
            {
                var term = new PolynomialTerm();
                var dividend = new List<PolynomialTerm>();
                string polynomialTernaryRepresentation = "";

                term.Power = i;
                term.Coefficient = 1;
                dividend.Add(term);

                var remainder = polynomialDivisor.PolynomDivider(dividend, divisor);

                foreach (var x in remainder)
                {
                    polynomialTernaryRepresentation += x.Coefficient.ToString();
                }

                GFElement currentGFElement = new GFElement
                {
                    AlphaPower = i,
                    NegativeAlphaPower = i - NumberOfGFElements,
                    TernaryRepresentation = polynomialTernaryRepresentation,
                    DecimalRepresentation =
Convert.ToInt32(numbersReformatting.ComposingStringFromListOfMultidigitalNumber(notati
onTranslation.TranslationFromNToM(polynomialTernaryRepresentation)))
                };

                gFElementsList.AddLast(currentGFElement);
                i++;
            }

            GFElement zeroGFElement = new GFElement
            {
                AlphaPower = int.MinValue,
                NegativeAlphaPower = int.MinValue,
```

```
                DecimalRepresentation = 0
            };

            for (int j = 1; j < polynomial.Length; j++)
zeroGFElement.TernaryRepresentation += "0";

            gFElementsList.AddFirst(zeroGFElement);

            GFElements = gFElementsList;
            DigitCapacity = polynomial.Length - 1;
        }

        public ComposingGF() { }

        private List<PolynomialTerm> DecodePolynomialNotation(string polynomialText)
        {
            var polynomial = new List<PolynomialTerm>();

            int powerCount = 1;
            foreach (var x in polynomialText)
            {
                var polynom = new PolynomialTerm
                {
                    Power = polynomialText.Length - powerCount,
                    Coefficient = Convert.ToInt32(char.GetNumericValue(x))
                };

                polynomial.Add(polynom);
                powerCount++;
            }

            return polynomial;
        }

        /// <summary>
        /// Отримання всіх поданнів одного елементу за його числовим або степеневим
представленням
        /// </summary>
        /// <param name="elementRepresentation"></param>
        /// <param name="representationType">"decimal" - за десятковим представленням,
"alphaPower" - за степенем альфа,
        /// "alphaPowerNegative" - за від’ємним степенем альфа</param>
        /// <returns></returns>
        public GFElement GetGFElement(int elementRepresentation, string
representationType)
        {
            foreach (var element in GFElements)
            {
                if (representationType == "decimal")
                {
                    if (elementRepresentation == element.DecimalRepresentation) return
element;
                }

                if (representationType == "alphaPower")
                {
                    if (elementRepresentation == element.AlphaPower) return element;
                }

                if (representationType == "alphaPowerNegative")
                {
```

```
                    if (elementRepresentation == element.NegativeAlphaPower) return
element;
                }
            }

            return null;
        }

        /// <summary>
        /// Отримання всіх поданнів одного елементу за його трійковим представленням
        /// </summary>
        /// <param name="ternaryRepresentation"></param>
        /// <returns></returns>
        public GFElement GetGFElement(string ternaryRepresentation)
        {
            foreach (var element in GFElements)
            {
                if (ternaryRepresentation == element.TernaryRepresentation) return
element;
            }

            return null;
        }

    }
}


using System.Collections.Generic;
using BGWCodeLibrary.ServingMethods;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.ServingMethods
{
    public class DataCodingSettings
    {
        public Alphabet alphabet;

        public int CorrectionLevel { get; set; } // lambda
        public int NumberOfControlCodewords { get; set; } // r
        public int NumberOfErrors { get; set; } // gamma
        public int NumberOfErasures { get; set; } // delta

        public PolynomialBracket GeneratorPolynomial { get; set; }

        public DataCodingSettings (Alphabet alphabet,
            int correctionLevel, int numberOfControlCodewords, int numberOfErrors, int
numberOfErasures)
        {
            this.alphabet = alphabet;

            CorrectionLevel = correctionLevel;
            NumberOfControlCodewords = numberOfControlCodewords;
            NumberOfErrors = numberOfErrors;
            NumberOfErasures = numberOfErasures;
        }

        public DataCodingSettings()
        {
        }
    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ReedSolomon.ServingMethods
{
    /// <summary>
    /// Отримання поліному з вхідної послідовності кодослів
    /// і навпаки
    /// </summary>
    class TranslatingBetweenPolynomialAndCodewords
    {
        private ComposingGF gfElements = new ComposingGF(); // поле Галуа

        public TranslatingBetweenPolynomialAndCodewords(ComposingGF gfElements)
        {
            this.gfElements = gfElements;
        }

        /// <summary>
        /// Отримання многочлену з послідовності кодослів
        /// </summary>
        /// <returns></returns>
        public PolynomialBracket ObtainPolynomialFromCodeword(LinkedList<int>
codeword) // codeword - вхідні дані у системі числення P_Omega
        {
            var xTerms = new List<PolynomialTerm>();
            var alphaTerms = new List<PolynomialTerm>();
            var xAlphaTerms = new List<PolynomialMixedTerm>();

            int maxPower = codeword.Count - 1;

            foreach (var code in codeword)
            {
                GFElement alpha = new GFElement();
                alpha = gfElements.GetGFElement(code, "decimal");

                if (alpha != null)
                {
                    if (maxPower != 0)
                    {
                        var polynomialTerm = new PolynomialMixedTerm()
                        {
                            Power = maxPower,
                            TermCoefficient = new PolynomialTerm()
                            {
                                Power = alpha.AlphaPower,
                                Coefficient = 1
                            }
                        };

                        xAlphaTerms.Add(polynomialTerm);
                    }
                    else
                    {
                        var polynomialTerm = new PolynomialTerm()
                        {
                            Power = alpha.AlphaPower,
                            Coefficient = 1
                        };
```

```
                    alphaTerms.Add(polynomialTerm);
                }
            }

            maxPower--;
        }

        var polynomial = new PolynomialBracket()
        {
            xTerms = xTerms,
            xAlphaTerms = xAlphaTerms,
            alphaTerms = alphaTerms
        };

        return polynomial;
    }

    /// <summary>
    /// Отримання послідовності кодослів з кодового многочлену
    /// </summary>
    /// <returns></returns>
    public LinkedList<int> ObtainCodewordFromPolynomial(PolynomialBracket
polynomial)
    {
        var xAlphaTerms = polynomial.xAlphaTerms;
        var alphaTerms = polynomial.alphaTerms;

        int maxPower = 0;
        foreach (var xAlpha in xAlphaTerms)
        {
            if (xAlpha.Power > maxPower) maxPower = xAlpha.Power;
        }

        int[] codewordList = new int[xAlphaTerms.Count + alphaTerms.Count];

        codewordList[codewordList.Length - 1] =
gfElements.GetGFElement(alphaTerms.First().Power, "alphaPower").DecimalRepresentation;
        foreach (var xAlpha in xAlphaTerms ?? new List<PolynomialMixedTerm>())
        {
            codewordList[codewordList.Length - xAlpha.Power - 1] =
gfElements.GetGFElement(xAlpha.TermCoefficient.Power,
"alphaPower").DecimalRepresentation;
        }

        var codewords = new LinkedList<int>();

        for (int i = 0; i < codewordList.Length; i++)
        {
            codewords.AddLast(codewordList[i]);
        }

        return codewords;
    }

    /// <summary>
    /// Отримання коефіцієнту альфа члену х многочлену з коефіцієнту спільного
множника х
    /// </summary>
    /// <returns></returns>
    public PolynomialTerm ObtainAlphaTermFromCommonCoefficient(string
commonCoefficient)
```

```
        {
            foreach (var element in gfElements.GFElements)
            {
                if (commonCoefficient == element.TernaryRepresentation)
                {
                    var term = new PolynomialTerm()
                    {
                        Power = element.AlphaPower,
                        Coefficient = 1
                    };

                    return term;
                }
            }

            return null;
        }
    }
}

using System;
using BGWCodeLibrary.ReedSolomon.ServingMethods;

namespace BGWCodeLibrary.ServingMethods.Maths
{
    class AlphaPowerCalculation
    {
        private ComposingGF composingGF;

        public AlphaPowerCalculation(ComposingGF composingGF)
        {
            this.composingGF = composingGF;
        }

        public int MultiplyAlphaTerms(int firstAlphaPower, int secondAlphaPower)
        {
            var alphaPower = (firstAlphaPower + secondAlphaPower) %
composingGF.NumberOfGFElements;

            return alphaPower < 0 ? composingGF.GetGFElement(alphaPower,
"alphaPowerNegative").AlphaPower : alphaPower;
        }

        public int DivideAlphaTerms(int firstAlphaPower, int secondAlphaPower)
        {
            var alphaPower = firstAlphaPower - secondAlphaPower;

            return alphaPower < 0 ? composingGF.GetGFElement(alphaPower,
"alphaPowerNegative").AlphaPower : alphaPower;
        }

        public int MultiplyAlphaTermWithCoefficient(int alphaPower, int coeff)
        {
            if (coeff == 1) { return alphaPower; }
            else
            {
                var alphaTernaryCode = alphaPower >= 0 ?
composingGF.GetGFElement(alphaPower, "alphaPower").TernaryRepresentation
                    : composingGF.GetGFElement(alphaPower,
"alphaPowerNegative").TernaryRepresentation;

                TernaryMaths ternaryOperations = new TernaryMaths();
```

```
                string ternaryCode = "";
                for (int i = 0; i < alphaTernaryCode.Length; i++)
                {
                    ternaryCode += ternaryOperations.TernaryMultiplication(coeff,
Int32.Parse(alphaTernaryCode[i].ToString())).ToString();
                }

                return composingGF.GetGFElement(ternaryCode).AlphaPower;
            }
        }

        public int SubstractAlphaTerms(int firstAlphaPower, int secondAlphaPower)
        {
            var ternaryCodes = TernaryCodesPreparation
                (firstAlphaPower % composingGF.NumberOfGFElements, secondAlphaPower %
composingGF.NumberOfGFElements);

            TernaryMaths ternaryOperations = new TernaryMaths();

            string ternaryCode = "";
            for (int i = 0; i < ternaryCodes.GetLength(1); i++)
            {
                ternaryCode += ternaryOperations.TernarySubstraction(ternaryCodes[0,
i], ternaryCodes[1, i]).ToString();
            }

            return composingGF.GetGFElement(ternaryCode).AlphaPower;
        }

        public int AddAlphaTerms(int firstAlphaPower, int secondAlphaPower)
        {
            var ternaryCodes = TernaryCodesPreparation(firstAlphaPower,
secondAlphaPower);

            TernaryMaths ternaryOperations = new TernaryMaths();

            string ternaryCode = "";
            for (int i = 0; i < ternaryCodes.GetLength(1); i++)
            {
                ternaryCode += ternaryOperations.TernaryAddition(ternaryCodes[0, i],
ternaryCodes[1, i]).ToString();
            }

            return composingGF.GetGFElement(ternaryCode).AlphaPower;
        }

        private int[,] TernaryCodesPreparation(int firstAlphaPower, int
secondAlphaPower)
        {
            var firstTernaryCode = firstAlphaPower >= 0 ?
composingGF.GetGFElement(firstAlphaPower, "alphaPower").TernaryRepresentation :
                composingGF.GetGFElement(firstAlphaPower,
"alphaPowerNegative").TernaryRepresentation; ;
            var secondTernaryCode = secondAlphaPower >= 0 ?
composingGF.GetGFElement(secondAlphaPower, "alphaPower").TernaryRepresentation :
                composingGF.GetGFElement(secondAlphaPower,
"alphaPowerNegative").TernaryRepresentation; ;

            int[,] ternaryCodes = new int[2, composingGF.DigitCapacity];

            int i = 0;
```

```
            foreach (var digit in firstTernaryCode)
            {
                ternaryCodes[0, i] = Int32.Parse(digit.ToString());
                i++;
            }

            i = 0;
            foreach (var digit in secondTernaryCode)
            {
                ternaryCodes[1, i] = Int32.Parse(digit.ToString());
                i++;
            }

            return ternaryCodes;
        }
    }
}


using System.Collections.Generic;
using BGWCodeLibrary.Exceptions;
using BGWCodeLibrary.ReedSolomon.ServingMethods;

namespace BGWCodeLibrary.ServingMethods.Maths
{
    public class MatrixDeterminant
    {
        /// <summary>
        /// Детермінант дорівнює альфа у степені (тут зберігається лише степінь)
        /// </summary>
        public int Determinant { get; set; }

        AlphaPowerCalculation alphaPowerCalculation;
        ServingMatrixMethods matrixMethods;
        ComposingGF composingGF;

        public MatrixDeterminant(ComposingGF composingGF)
        {
            this.composingGF = composingGF;
            alphaPowerCalculation = new AlphaPowerCalculation(composingGF);
        }

        /// <summary>
        /// Обчислення детермінанту вхідної матриці
        /// </summary>
        /// <param name="matrix"></param>
        public void GetMatrixDeterminant(int[,] matrix)
        {
            var alphaTermsInDeterminant = ObtainMatrixDeterminant(matrix);

            var cummulativeAlphaTerm = alphaTermsInDeterminant[0];
            for (int i = 1; i < alphaTermsInDeterminant.Count; i++)
            {
                cummulativeAlphaTerm =
alphaPowerCalculation.AddAlphaTerms(cummulativeAlphaTerm, alphaTermsInDeterminant[i]);
            }

            Determinant = cummulativeAlphaTerm;
        }

        private List<int> ObtainMatrixDeterminant(int[,] matrix)
        {
```

```
            int rows = matrix.GetLength(0);
            int colomns = matrix.GetLength(1);

            List<int> determinantComponents = new List<int>();

            if (rows != colomns) throw new WrongSizeOfMatrixException("Matrix is not
square.");

            if (rows == 0) throw new WrongSizeOfMatrixException("Matrix is empty.");

            if (rows == 1) determinantComponents.Add(matrix[0, 0]);

            if (rows == 2)
determinantComponents.Add(alphaPowerCalculation.SubstractAlphaTerms(
                alphaPowerCalculation.MultiplyAlphaTerms(matrix[0, 0], matrix[1, 1]),
                alphaPowerCalculation.MultiplyAlphaTerms(matrix[0, 1], matrix[1,
0])));

            if (rows >= 3)
            {
                matrixMethods = new ServingMatrixMethods(composingGF);

                for (int j = 0; j < rows; j++)
                {
                    int coeff = (1 + j) % 2 == 0 ? 1 : 2; // -1 = 2
                    GetMatrixDeterminant(matrixMethods.ObtainMinorMatrix(matrix, 0,
j));


determinantComponents.Add(alphaPowerCalculation.MultiplyAlphaTermWithCoefficient
                        (alphaPowerCalculation.MultiplyAlphaTerms(matrix[0, j],
Determinant), coeff));
                }
            }

            return determinantComponents;
        }
    }
}


using System.Collections.Generic;
using BGWCodeLibrary.ReedSolomon.ServingMethods;

namespace BGWCodeLibrary.ServingMethods.Maths
{
    class MatrixMultiplication
    {
        AlphaPowerCalculation alphaPowerCalculation;

        public MatrixMultiplication(ComposingGF composingGF)
        {
            alphaPowerCalculation = new AlphaPowerCalculation(composingGF);
        }

        public List<int> MultiplyMatrixByVector(int[,] matrix, int[] matrixVector)
        {
            int rows = matrix.GetLength(0);
            int colomns = matrix.GetLength(1);

            var resultVector = new List<int>();
```

```
            int midRes = int.MinValue;
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < colomns; j++)
                {
                    midRes = alphaPowerCalculation.AddAlphaTerms(midRes,
                        alphaPowerCalculation.MultiplyAlphaTerms(matrix[i, j],
matrixVector[j]));
                }
                resultVector.Add(midRes);
            }

            return resultVector;
        }
    }
}


using BGWCodeLibrary.Exceptions;
using BGWCodeLibrary.ReedSolomon.ServingMethods;

namespace BGWCodeLibrary.ServingMethods.Maths
{
    class ServingMatrixMethods
    {
        ComposingGF composingGF;
        AlphaPowerCalculation alphaPowerCalculation;
        MatrixDeterminant obtainDeterminant;

        public ServingMatrixMethods(ComposingGF composingGF)
        {
            this.composingGF = composingGF;
            alphaPowerCalculation = new AlphaPowerCalculation(composingGF);
            obtainDeterminant = new MatrixDeterminant(composingGF);
        }

        /// <summary>
        /// Обчислення мінору вхідної матриці за викреслених рядка та стовпця
        /// </summary>
        /// <param name="currentMatrix"></param>
        /// <param name="crossedRow"></param>
        /// <param name="crossedColomn"></param>
        /// <returns></returns>
        public int[,] ObtainMinorMatrix(int[,] currentMatrix, int crossedRow, int
crossedColomn)
        {
            int[,] minorMatrix = new int[currentMatrix.GetLength(0) - 1,
currentMatrix.GetLength(1) - 1];

            for (int i = 0, ii = 0; i < currentMatrix.GetLength(0) && ii <
minorMatrix.GetLength(0); i++)
            {
                bool flag = false;
                for (int j = 0, jj = 0; j < currentMatrix.GetLength(1) && jj <
minorMatrix.GetLength(1); j++)
                {
                    if (i != crossedRow && j != crossedColomn)
                    {
                        minorMatrix[ii, jj] = currentMatrix[i, j];
                        flag = true;
                        jj++;
                    }
```

```
                }
                if (flag) ii++;
            }

            return minorMatrix;
        }

        /// <summary>
        /// Обчислення союзної матриці
        /// </summary>
        /// <param name="matrix"></param>
        /// <returns></returns>
        private int[,] ObtainAdjugateMatrix(int[,] matrix)
        {
            var cofactorMatrix = ObtainCofactorMatrix(matrix);
            return MatrixTransposition(cofactorMatrix);
        }

        /// <summary>
        /// Обчислення оберненої матриці
        /// </summary>
        /// <param name="matrix"></param>
        /// <returns></returns>
        public int[,] ObtainInverseMatrix(int[,] matrix)
        {
            if (matrix.GetLength(0) == 0) throw new WrongSizeOfMatrixException("Empty
matrix");

            if (matrix.GetLength(0) == 1)
            {
                matrix[0, 0] = composingGF.GetGFElement(matrix[0, 0] * (-1),
"alphaPowerNegative").AlphaPower;
                return matrix;
            }

            var adjugateMatrix = ObtainAdjugateMatrix(matrix);

            int[,] inverseMatrix = new int[matrix.GetLength(0), matrix.GetLength(1)];

            obtainDeterminant.GetMatrixDeterminant(matrix);
            var determinant = obtainDeterminant.Determinant;

            var invertedDeterminant = composingGF.GetGFElement(determinant * (-1),
"alphaPowerNegative").AlphaPower;

            for (int i = 0; i < adjugateMatrix.GetLength(0); i++)
            {
                for (int j = 0; j < adjugateMatrix.GetLength(1); j++)
                {
                    inverseMatrix[i, j] =
alphaPowerCalculation.MultiplyAlphaTerms(invertedDeterminant, adjugateMatrix[i, j]);
                }
            }

            return inverseMatrix;
        }

        /// <summary>
        /// Обчислення матриці алгебраїчних доповнень
        /// </summary>
        /// <param name="matrix"></param>
        /// <returns></returns>
```

```
        private int[,] ObtainCofactorMatrix(int[,] matrix)
        {
            int[,] cofactorMatrix = new int[matrix.GetLength(0), matrix.GetLength(1)];

            for (int i = 0; i < cofactorMatrix.GetLength(0); i++)
            {
                for (int j = 0; j < cofactorMatrix.GetLength(1); j++)
                {
                    int coeff = (i + j) % 2 == 0 ? 1 : 2; // -1 = 2
                    var minorMatrix = ObtainMinorMatrix(matrix, i, j);

                    obtainDeterminant.GetMatrixDeterminant(minorMatrix);
                    var determinator = obtainDeterminant.Determinant;

                    cofactorMatrix[i, j] =
alphaPowerCalculation.MultiplyAlphaTermWithCoefficient(determinator, coeff);
                }
            }

            return cofactorMatrix;
        }

        /// <summary>
        /// Обчислення транспонованої матриці
        /// </summary>
        /// <param name="matrix"></param>
        /// <returns></returns>
        private int[,] MatrixTransposition(int[,] matrix)
        {
            int[,] transposedMatrix = new int[matrix.GetLength(0),
matrix.GetLength(1)];

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1); j++)
                {
                    transposedMatrix[i, j] = matrix[j, i];
                }
            }

            return transposedMatrix;
        }
    }
}


using System.Linq;
using static BGWCodeLibrary.ReedSolomon.PolynomialProcessing.PolynomialStructure;

namespace BGWCodeLibrary.ServingMethods.Maths
{
    class SolveEquation
    {
        public int LinearEquation(PolynomialBracket polynomial)
        {
            return 364 - polynomial.xAlphaTerms.First().TermCoefficient.Power; // 2 /
alpha^Power = alpha^364 / alpha^Power;
        }
    }
}
```

```
namespace BGWCodeLibrary.ServingMethods
{
    class TernaryMaths
    {
        public int TernaryAddition(int a, int b)
        {
            if (a == 0 && b == 0)
            {
                return 0;
            }
            else

            if ((a == 0 && b == 1) || (a == 1 && b == 0))
            {
                return 1;
            }
            else

            if ((a == 0 && b == 2) || (a == 2 && b == 0))
            {
                return 2;
            }
            else

            if (a == 1 && b == 1)
            {
                return 2;
            }
            else

            if ((a == 1 && b == 2) || (a == 2 && b == 1))
            {
                return 0;
            }
            else

            if (a == 2 && b == 2)
            {
                return 1;
            }
            else
                return -1;
        }

        public int TernarySubstraction(int a, int b)
        {
            if (a == 0 && b == 0)
            {
                return 0;
            }
            else

            if (a == 0 && b == 1)
            {
                return 2;
            }
            else

            if (a == 0 && b == 2)
            {
                return 1;
            }
```

```
            else

            if (a == 1 && b == 0)
            {
                return 1;
            }
            else

            if (a == 1 && b == 1)
            {
                return 0;
            }
            else

            if (a == 1 && b == 2)
            {
                return 2;
            }
            else

            if (a == 2 && b == 0)
            {
                return 2;
            }
            else

            if (a == 2 && b == 1)
            {
                return 1;
            }
            else

            if (a == 2 && b == 2)
            {
                return 0;
            }
            else
                return -1;
        }

        public int TernaryMultiplication(int a, int b)
        {
            if ((a == 0 && b == 0) || (a == 0 && b == 1) || (a == 1 && b == 0) || (a
== 0 && b == 2) || (a == 2 && b == 0))
            {
                return 0;
            }
            else

            if ((a == 1 && b == 1) || (a == 2 && b == 2))
            {
                return 1;
            }
            else

            if ((a == 1 && b == 2) || (a == 2 && b == 1))
            {
                return 2;
            }
            else
                return -1;
        }
```

```
    }
}

namespace BGWCodeLibrary.ServingMethods.Parsers
{
    public abstract class BaseAlphabetParser
    {
        protected AlphabetElement DecomposeAlphabetElement(string stringToDecompose)
        {
            string element, elementCode;
            Decompose(stringToDecompose, out element, out elementCode);
            return new AlphabetElement(element, int.Parse(elementCode));
        }

        // decomposition to string
        protected string DecomposeElement(string stringToDecompose)
        {
            string element;
            Decompose(stringToDecompose, out element, out _);
            return element;
        }

        private static void Decompose(string stringToDecompose, out string element,
out string elementCode)
        {
            element = "";
            elementCode = "";
            int i = 0;
            while (stringToDecompose[i] != ' ')
            {
                elementCode += stringToDecompose[i];
                i++;
            }

            i++;
            while (i < stringToDecompose.Length)
            {
                element += stringToDecompose[i];
                i++;
            }

            if (element == "space")
            {
                element = " ";
            }
        }
    }
}


using System.Collections.Generic;
using System.IO;

namespace BGWCodeLibrary.ServingMethods.Parsers
{
    public class BytesParser : BaseAlphabetParser
    {
        public Alphabet Parse(byte[] content)
        {
            var alphabetList = new List<AlphabetElement>();
            var rawFileString = System.Text.Encoding.Default.GetString(content);
            using (var stream = new StringReader(rawFileString))
```

```
            {
                string bufLine;
                while ((bufLine = stream.ReadLine()) != null)
                {
                    alphabetList.Add(DecomposeAlphabetElement(bufLine));
                }
            }
            return new Alphabet(alphabetList);
        }
    }
}


using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace BGWCodeLibrary.ServingMethods
{
    public class Alphabet : IEnumerable<AlphabetElement>
    {
        private List<AlphabetElement> _letters { get; set; }

        public Alphabet(List<AlphabetElement> elements)
        {
            _letters = elements;
        }

        public int Count => _letters.Count;

        public IEnumerable<AlphabetElement> Letters => _letters;

        public IEnumerable<string> Elements => _letters.Select(x => x.Element);

        public AlphabetElement this[int index]
        {
            get => _letters[index];
            private set => _letters[index] = value;
        }

        #region IEnumerable
        public IEnumerator<AlphabetElement> GetEnumerator()
        {
            return _letters.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return _letters.GetEnumerator();
        }
        #endregion
    }
}

namespace BGWCodeLibrary.ServingMethods
{
    /// <summary>
    /// Один елемент алфавіту, що складається з самого елементу та його коду -
    /// порядкового номера цього елемента в алфавіті
    /// </summary>
    public class AlphabetElement
    {
```

```
        public int Code { get; }
        public string Element { get; }

        public AlphabetElement(string alphabetElement, int alphabetElementCode)
        {
            Element = alphabetElement;
            Code = alphabetElementCode;
        }
    }
}

using System.Collections.Generic;
using System.Linq;

namespace BGWCodeLibrary.ServingMethods
{
    class NumbersReformatting
    {
        private readonly char[] digitsArray = { ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’,
‘8’, ‘9’ };

        /* ------------------- РОБОТА З ЧИСЛАМИ ------------------- */

        /// <summary>
        /// Розбиття рядка, яким представлене число, на окремі цифри певної системи
числення
        /// </summary>
        /// <param name="number"></param>
        /// <returns></returns>
        public LinkedList<int> DecompositionOfMultidigitNumberToDigits(string number)
        {
            var decomposedDigits = new LinkedList<int>();

            bool beginningofNumber = false;
            bool endOfNumber = false;

            string multidigitNumber = "";

            for (int i = 0; i < number.Length; i++)
            {
                if (digitsArray.Contains(number[i]) && !beginningofNumber)
                {
                    decomposedDigits.AddLast(int.Parse(number[i].ToString()));
                }

                if (number[i] == ‘\’’ && !beginningofNumber)
                {
                    beginningofNumber = true;
                }
                else if (number[i] == ‘\’’ && beginningofNumber)
                {
                    beginningofNumber = false;
                    endOfNumber = true;
                }

                if (number[i] != ‘\’’ && beginningofNumber)
                {
                    multidigitNumber += number[i];
                }

                if (endOfNumber)
                {
```

```
                    endOfNumber = false;
                    decomposedDigits.AddLast(int.Parse(multidigitNumber));
                    multidigitNumber = "";
                }
            }

            return decomposedDigits;
        }

        /// <summary>
        /// Формування рядка зі списку окремих цифр, з яких складається число у певній
системі числення
        /// </summary>
        /// <param name="numberToCompose"></param>
        /// <returns></returns>
        public string ComposingStringFromListOfMultidigitalNumber(LinkedList<int>
numberToCompose)
        {
            string composedString = "";

            foreach (var digit in numberToCompose)
            {
                if (digit > 9)
                {
                    composedString += '\'' + digit.ToString() + '\'';
                }
                else
                {
                    composedString += digit.ToString();
                }
            }

            return composedString;
        }
    }
}

using System.Drawing;

namespace BGWCodeLibrary.Vizualization.BarcodeStructure
{
    public class Cell
    {
        public int x0; // start X-coordinate for a cell
        public int y0; // start Y-coordinate for a cell
        public int edgeLength; // length of a cell edge
        public CellColors color; // color of a cell

        public Cell(int x, int y, int length, CellColors color)
        {
            x0 = x;
            y0 = y;
            edgeLength = length;
            this.color = color;
        }

        public void Draw(Graphics drawingArea)
        {
            Pen p = new Pen(SetCellColor());
            SolidBrush sb = new SolidBrush(SetCellColor());

            drawingArea.DrawRectangle(p, x0, y0, edgeLength, edgeLength);
```

```
            drawingArea.FillRectangle(sb, x0, y0, edgeLength, edgeLength);
        }

        private Color SetCellColor()
        {
            Color cellColor = Color.Red;

            if (color == CellColors.BLACK)
            {
                cellColor = Color.Black;
            }
            else if (color == CellColors.GRAY)
            {
                cellColor = Color.Gray;
            }
            else if (color == CellColors.WHITE)
            {
                cellColor = Color.White;
            }

            return cellColor;
        }
    }
}


namespace BGWCodeLibrary.Vizualization.BarcodeStructure
{
    public enum CellColors
    {
        BLACK,
        GRAY,
        WHITE,
        UNKNOWN
    }
}

using System;
using BGWCodeLibrary.Vizualization.BarcodeSymbolForming;

namespace BGWCodeLibrary.Vizualization.BarcodeStructure
{
    public class Matrix
    {
        public int[,] BarcodeMatrix { get; private set; }
        public int NumberOfPatterns { get; private set; }
        public int Height { get; private set; }
        public int Width { get; private set; }
        public int PatternHeight { get; private set; }
        public int PatternWidth { get; private set; }
        public int DigitCapacity { get; private set; }
        public PatternFormat PatternFormat { get; private set; }

        public Matrix(int digitCapacity, int numberOfPatterns, PatternFormat
patternFormat)
        {
            DigitCapacity = digitCapacity;
            NumberOfPatterns = numberOfPatterns;
            PatternFormat = patternFormat;

            CalculateMatrixSize(numberOfPatterns);
            BarcodeMatrix = new int[Height, Width];
```

```
}

/// <summary>
/// Calculates dimensions of a barcode matrix for the defined digit capacity
/// </summary>
public void CalculateMatrixSize(int numberOfPatterns)
{
    double idealNumber = numberOfPatterns / 2 + 1;
    int firstSide = (int)Math.Ceiling(Math.Sqrt(idealNumber));
    int secondSide = (int)Math.Ceiling(idealNumber / firstSide);

    Height = firstSide > secondSide ? secondSide : firstSide;
    Width = firstSide > secondSide ? firstSide : secondSide;

    GetSizeOfPattern();

    Height++;
    Width++;
}

/// <summary>
/// Gets the size of a pattern for the digit capacity defined in a class
/// </summary>
public void GetSizeOfPattern()
{
    switch (DigitCapacity)
    {
        case 4:
            PatternHeight = PatternWidth = 2;
            break;
        case 5:
            if (PatternFormat == PatternFormat.NM_RECTANGLE_LADDER)
            {
                PatternHeight = 2; PatternWidth = 3;
            }
            else
            {
                PatternHeight = 3; PatternWidth = 2;
            }
            break;
        case 6:
            PatternHeight = PatternWidth = 3;
            Height *= PatternHeight;
            Width *= PatternWidth + 1;
            break;
        case 7:
            if (PatternFormat == PatternFormat.NM_RECTANGLE_LADDER)
            {
                PatternHeight = 2; PatternWidth = 4;
            }
            else
            {
                PatternHeight = 4; PatternWidth = 2;
            }
            break;
        case 8:
            if (PatternFormat == PatternFormat.NM_RECTANGLE)
            {
                PatternHeight = 2; PatternWidth = 4;
            }
            else
            {
```

```
                        PatternHeight = 4; PatternWidth = 2;
                    }
                    break;
                case 9:
                    PatternHeight = PatternWidth = 3;
                    break;
                case 10:
                    PatternHeight = PatternWidth = 4;
                    break;
            }
        }
    }
}


using System.Collections.Generic;
using System.Drawing;

namespace BGWCodeLibrary.Vizualization.BarcodeStructure
{
    public class Symbol
    {
        private int x0, y0;
        private readonly int cellLength;

        private readonly Matrix matrix;
        public LinkedList<Cell> BarcodeCellCoordinates { get; private set; } = new
LinkedList<Cell>();

        public Symbol(Matrix matrix, int x0, int y0, int cellLength)
        {
            this.matrix = matrix;
            this.x0 = x0; this.y0 = y0;
            this.cellLength = cellLength;

            BuildBarcodeSymbol();
        }

        private void BuildBarcodeSymbol()
        {
            Cell cell;
            int x = x0;
            int y = y0;

            for (int i = 0; i < matrix.Height; i++)
            {
                for (int j = 0; j < matrix.Width; j++)
                {
                    CellColors cellColor =
                        matrix.BarcodeMatrix[i, j] == 0 ? CellColors.WHITE :
                        matrix.BarcodeMatrix[i, j] == 1 ? CellColors.GRAY :
CellColors.BLACK;
                    cell = new Cell(x, y, cellLength, cellColor);
                    BarcodeCellCoordinates.AddLast(cell);
                    x = cell.x0 + cellLength;
                }
                x = x0;
                y = BarcodeCellCoordinates.Last.Value.y0 + cellLength;
            }
        }

        public void Draw(Graphics drawingArea)
```

```
        {
            foreach (var cell in BarcodeCellCoordinates)
            {
                cell.Draw(drawingArea);
            }
        }
    }
}


using System;
using BGWCodeLibrary.DataEncoding;
using BGWCodeLibrary.Vizualization.BarcodeStructure;
using Nito.Collections;

namespace BGWCodeLibrary.Vizualization.BarcodeSymbolForming
{
    class BarcodeMatrixBuilder
    {
        public Matrix BarcodeMatrix { get; private set; }

        public BarcodeMatrixBuilder(int digitCapacity, TernaryCode ternaryCode,
PatternFormat patternFormat)
        {
            BuildMatrix(digitCapacity, ternaryCode, patternFormat);
        }

        /// <summary>
        /// For digit capacity equal to 6 or 10
        /// </summary>
        private void BuildMatrix(int digitCapacity, TernaryCode ternaryCode,
PatternFormat patternFormat)
        {
            var ternarySymbol = ternaryCode.TernarySymbol;
            Deque<Deque<int>> deque = new Deque<Deque<int>>(ternarySymbol.Count);

            BarcodeMatrix = new Matrix(digitCapacity, ternarySymbol.Count,
patternFormat);

            foreach (var pattern in ternarySymbol)
            {
                deque.AddToBack(pattern.TernarySequence);
            }

            BarcodeFrame(BarcodeMatrix);

            int iStart = 1;
            int jStart = 1;

            int checkSum = 1;
            int numberOfQuadrants = 0;

            while (deque.Count != 0)
            {
                numberOfQuadrants++;

                Deque<int> patternOne = deque.RemoveFromFront();
                Deque<int> patternTwo = deque.Count != 0 ? deque.RemoveFromBack() :
null;

                int iCount = 0;
                int jCount = 0;
```

```
                for (int i = iStart; iCount < BarcodeMatrix.PatternHeight; i++,
iCount++)
                {
                    for (int j = jStart; jCount < BarcodeMatrix.PatternWidth; j++,
jCount++)
                    {
                        BarcodeMatrix.BarcodeMatrix[i, j] =
patternOne.RemoveFromFront();
                        if (CheckDiagonalSum(i, j, numberOfQuadrants) == checkSum)
                        {
                            checkSum++;
                            break;
                        }
                    }
                    jCount = 0;
                }

                int k = jStart + 1;
                iCount = 0;

                for (int i = iStart; iCount < BarcodeMatrix.PatternHeight; i++,
iCount++)
                {
                    for (int j = k; j <= jStart + BarcodeMatrix.PatternWidth; j++)
                    {
                        if (patternTwo != null)
                        {
                            BarcodeMatrix.BarcodeMatrix[i, j] =
patternTwo.RemoveFromBack();
                        }
                        else
                        {
                            BarcodeMatrix.BarcodeMatrix[i, j] = 0;
                        }
                    }
                    k++;
                }

                if (jStart + BarcodeMatrix.PatternWidth + 1 >= BarcodeMatrix.Width)
                {
                    iStart += BarcodeMatrix.PatternHeight;
                    jStart = 1;
                    checkSum = 1;
                }
                else
                {
                    jStart += BarcodeMatrix.PatternWidth + 1;
                    checkSum = 1;
                }
            }
        }

        private int CheckDiagonalSum(int i, int j, int numberOfQuadrants)
        {
            if (numberOfQuadrants > 1)
            {
                while (i > BarcodeMatrix.PatternHeight)
                {
                    i -= BarcodeMatrix.PatternHeight;
                }
                while (j > BarcodeMatrix.PatternWidth)
```

```
                {
                    j -= (BarcodeMatrix.PatternWidth + 1);
                }
            }

            return (i + j) / 2;
        }

        private void BarcodeFrame(Matrix matrix)
        {
            matrix.BarcodeMatrix[0, 0] = 2;
            matrix.BarcodeMatrix[0, 1] = 2;
            matrix.BarcodeMatrix[1, 0] = 2;

            for (int i = 2; i < matrix.Height; i++)
            {
                if (i % 2 == 0)
                {
                    matrix.BarcodeMatrix[i, 0] = 0;
                }
                else
                {
                    matrix.BarcodeMatrix[i, 0] = 2;
                }
            }

            for (int j = 2; j < matrix.Width; j++)
            {
                if (j % 2 == 0)
                {
                    matrix.BarcodeMatrix[0, j] = 0;
                }
                else
                {
                    matrix.BarcodeMatrix[0, j] = 2;
                }
            }

            for (int i = matrix.Height - matrix.PatternHeight; i < matrix.Height; i++)
            {
                for (int j = matrix.Width - matrix.PatternWidth - 1; j < matrix.Width;
j++)
                {
                    matrix.BarcodeMatrix[i, j] = 2;
                }
            }
        }
    }
}


using System.Drawing;
using BGWCodeLibrary.Vizualization.BarcodeStructure;

namespace BGWCodeLibrary.Vizualization.BarcodeSymbolForming
{
    public class DrawBarcode
    {
        private Matrix barcodeMatrix;
        Graphics drawingArea;

        public DrawBarcode(Matrix barcodeMatrix, Graphics drawingArea)
```

```
        {
            this.barcodeMatrix = barcodeMatrix;
            this.drawingArea = drawingArea;
        }

        private CellColors GetSetColor(int bit)
        {
            CellColors color = CellColors.UNKNOWN;

            if (bit == 0)
            {
                color = CellColors.WHITE;
            }
            else if (bit == 1)
            {
                color = CellColors.GRAY;
            }
            else if (bit == 2)
            {
                color = CellColors.BLACK;
            }

            return color;
        }

        public void Draw(Graphics drawingArea)
        {

        }
    }
}


namespace BGWCodeLibrary.Vizualization.BarcodeSymbolForming
{
    /// <summary>
    /// Defines shape of a pattern
    /// </summary>
    public enum PatternFormat
    {
        /// <summary>
        /// Digit capacity equals to 4 or 9
        /// </summary>
        SQUARE,

        /// <summary>
        /// Digit capacity equals to 6 or 8 (hozirontal rectangle)
        /// </summary>
        NM_RECTANGLE,
        /// <summary>
        /// Digit capacity equals to 6 or 8 (vertical rectangle)
        /// </summary>
        MN_RECTANGLE,

        /// <summary>
        /// Digit capacity equals to 5 or 7 (hozirontal rectangle)
        /// </summary>
        NM_RECTANGLE_LADDER,

        /// <summary>
        /// Digit capacity equals to 5 or 7 (vertical rectangle)
        /// </summary>
```

```
        MN_RECTANGLE_LADDER,

        /// <summary>
        /// Digit capacity equals to 6 or 10
        /// </summary>
        LADDER
    }
}
```

Додаток Е. Фрагменти програмного коду мовою C# для подання ДДШК

```
class UpperLayerPattern {
        public TernaryPattern TernaryPattern { get; private set; }
        public int UpperDigitCapacity { get; private set; }
        readonly int digitCapacity;
        public int ControlBit { get; private set; }
        readonly int upperTernaryDigit;
        int actualDigitColor;
        bool isEqual;

        public UpperLayerPattern(int digitCapacity, int controlBit, TernaryPattern
ternaryPattern)
{               this.digitCapacity = digitCapacity;
                UpperDigitCapacity = this.digitCapacity + 1;
                TernaryPattern = ternaryPattern;
                ControlBit = controlBit;
                GetPatternDigitColor();     }

        private void GetPatternDigitColor()
{       // визначення цифрового еквіваленту кольору ШК-знаку    }
        private void SetControlBit()
{       // встановлення контрольного біту  }

        public void SetUpperPattern()
{       // застосування контрольного біту до ШК-знаку нижнього рівня  }

        public void GetInitialPattern()
{       // отримання початкового ШК-знаку нижнього рівня}

}
```

Додаток Є. Завадостійке штрихове кодування

Таблиця Є.1. Коректувальні можливості коду Ріда-Соломона

Рівень корекції, Λ	Кількість контрольних ШК-знаків, r	Коректувальні можливості	
		Кількість помилок, γ	Кількість стирань, δ
0	0	-	-
1	1	0	1
2	3	1 0	1 3
3	7	3 2 1 0	1 3 5 7
4	15	7 6 5 4 3 2 1 0	1 3 5 7 9 11 13 15

Таблиця Є.2. Рекомендовані рівні корекції спотворень в ШК-позначках

Кількість інформаційних ШК-знаків, k	Рівень корекції, Λ
до 50	1
51-100	2
101-180	3

181-270	4
271-370	5
371-720	6
721-1000	7
понад 1000	8

Таблиця Є.3. Подання елементів поля $GF(3^6)$ за модулем незвідного поліному $m_6(x) = x^6 + x + 2$

Степінь α невід'ємний	**Степінь α від'ємний**	**Трійкова послідовність**	**Десяткове число**
-	-	000000	0
0	-728	000001	1
1	-727	000010	3
2	-726	000100	9
3	-725	001000	27
4	-724	010000	81
5	-723	100000	243
6	-722	000021	7
7	-721	000210	21
8	-720	002100	63
9	-719	021000	189
10	-718	210000	567
11	-717	100012	248
12	-716	000111	13
13	-715	001110	39
14	-714	011100	117
15	-713	111000	351
16	-712	110021	331
17	-711	100201	262
18	-710	002001	55
19	-709	020010	165
20	-708	200100	495
21	-707	001012	32
22	-706	010120	96
23	-705	101200	288
24	-704	012021	142
25	-703	120210	426

26	-702	202121	556
27	-701	021222	215
28	-700	212220	645
29	-699	122212	482
30	-698	222111	715
31	-697	221122	692
32	-696	211202	614
33	-695	112002	380
34	-694	120011	409
35	-693	200101	496
36	-692	001022	35
37	-691	010220	105
38	-690	102200	315
39	-689	022021	223
40	-688	220210	669
41	-687	202112	554
42	-686	021102	200
43	-685	211020	600
44	-684	110212	347
45	-683	102111	310
46	-682	021101	199
47	-681	211010	597
48	-680	110112	338
49	-679	101111	283
50	-678	011101	118
51	-677	111010	354
52	-676	110121	340
53	-675	101201	289
54	-674	012001	136
55	-673	120010	408
56	-672	200121	502
57	-671	001222	53
58	-670	012220	159
59	-669	122200	477
60	-668	222021	709
61	-667	220222	674
62	-666	202202	560
63	-665	022002	218
64	-664	220020	654
65	-663	200212	509
66	-662	002102	65
67	-661	021020	195
68	-660	210200	585
69	-659	102012	302
70	-658	020111	175

71	-657	201110	525
72	-656	011112	122
73	-655	111120	366
74	-654	111221	376
75	-653	112201	397
76	-652	122001	460
77	-651	220001	649
78	-650	200022	494
79	-649	000202	20
80	-648	002020	60
81	-647	020200	180
82	-646	202000	540
83	-645	020012	167
84	-644	200120	501
85	-643	001212	50
86	-642	012120	150
87	-641	121200	450
88	-640	212021	628
89	-639	120222	431
90	-638	202211	562
91	-637	022122	233
92	-636	221220	699
93	-635	212212	644
94	-634	122102	470
95	-633	221011	679
96	-632	210122	584
97	-631	101202	290
98	-630	012011	139
99	-629	120110	417
100	-628	201121	529
101	-627	011222	134
102	-626	112220	402
103	-625	122221	484
104	-624	222201	721
105	-623	222022	710
106	-622	220202	668
107	-621	202002	542
108	-620	020002	164
109	-619	200020	492
110	-618	000212	23
111	-617	002120	69
112	-616	021200	207
113	-615	212000	621
114	-614	120012	410
115	-613	200111	499

116	-612	001122	44
117	-611	011220	132
118	-610	112200	396
119	-609	122021	466
120	-608	220201	667
121	-607	202022	548
122	-606	020202	182
123	-605	202020	546
124	-604	020212	185
125	-603	202120	555
126	-602	021212	212
127	-601	212120	636
128	-600	121212	455
129	-599	212111	634
130	-598	121122	449
131	-597	211211	616
132	-596	112122	395
133	-595	121211	454
134	-594	212101	631
135	-593	121022	440
136	-592	210211	589
137	-591	102122	314
138	-590	021211	211
139	-589	212110	633
140	-588	121112	446
141	-587	211111	607
142	-586	111122	368
143	-585	111211	373
144	-584	112101	388
145	-583	121001	433
146	-582	210001	568
147	-581	100022	251
148	-580	000211	22
149	-579	002110	66
150	-578	021100	198
151	-577	211000	594
152	-576	110012	329
153	-575	100111	256
154	-574	001101	37
155	-573	011010	111
156	-572	110100	333
157	-571	101021	277
158	-570	010201	100
159	-569	102010	300
160	-568	020121	178

161	-567	201210	534
162	-566	012112	149
163	-565	121120	447
164	-564	211221	619
165	-563	112222	404
166	-562	122211	481
167	-561	222101	712
168	-560	221022	683
169	-559	210202	587
170	-558	102002	299
171	-557	020011	166
172	-556	200110	498
173	-555	001112	41
174	-554	011120	123
175	-553	111200	369
176	-552	112021	385
177	-551	120201	424
178	-550	202001	541
179	-549	020022	170
180	-548	200220	510
181	-547	002212	77
182	-546	022120	231
183	-545	221200	693
184	-544	212012	626
185	-543	120102	416
186	-542	201011	517
187	-541	010122	98
188	-540	101220	294
189	-539	012221	160
190	-538	122210	480
191	-537	222121	718
192	-536	221222	701
193	-535	212202	641
194	-534	122002	461
195	-533	220011	652
196	-532	200122	503
197	-531	001202	47
198	-530	012020	141
199	-529	120200	423
200	-528	202021	547
201	-527	020222	188
202	-526	202220	564
203	-525	022212	239
204	-524	222120	717
205	-523	221212	698

206	-522	212102	632
207	-521	121002	434
208	-520	210011	571
209	-519	100122	260
210	-518	001211	49
211	-517	012110	147
212	-516	121100	441
213	-515	211021	601
214	-514	110222	350
215	-513	102211	319
216	-512	022101	226
217	-511	221010	678
218	-510	210112	581
219	-509	101102	281
220	-508	011011	112
221	-507	110110	336
222	-506	101121	286
223	-505	011201	127
224	-504	112010	381
225	-503	120121	421
226	-502	201201	532
227	-501	012022	143
228	-500	120220	429
229	-499	202221	565
230	-498	022222	242
231	-497	222220	726
232	-496	222212	725
233	-495	222102	713
234	-494	221002	677
235	-493	210002	569
236	-492	100002	245
237	-491	000011	4
238	-490	000110	12
239	-489	001100	36
240	-488	011000	108
241	-487	110000	324
242	-486	100021	250
243	-485	000201	19
244	-484	002010	57
245	-483	020100	171
246	-482	201000	513
247	-481	010012	86
248	-480	100120	258
249	-479	001221	52
250	-478	012210	156

251	-477	122100	468
252	-476	221021	682
253	-475	210222	593
254	-474	102202	317
255	-473	022011	220
256	-472	220110	660
257	-471	201112	527
258	-470	011102	119
259	-469	111020	357
260	-468	110221	349
261	-467	102201	316
262	-466	022001	217
263	-465	220010	651
264	-464	200112	500
265	-463	001102	38
266	-462	011020	114
267	-461	110200	342
268	-460	102021	304
269	-459	020201	181
270	-458	202010	543
271	-457	020112	176
272	-456	201120	528
273	-455	011212	131
274	-454	112120	393
275	-453	121221	457
276	-452	212201	640
277	-451	122022	467
278	-450	220211	670
279	-449	202122	557
280	-448	021202	209
281	-447	212020	627
282	-446	120212	428
283	-445	202111	553
284	-444	021122	206
285	-443	211220	618
286	-442	112212	401
287	-441	122111	472
288	-440	221101	685
289	-439	211022	602
290	-438	110202	344
291	-437	102011	301
292	-436	020101	172
293	-435	201010	516
294	-434	010112	95
295	-433	101120	285

296	-432	011221	133
297	-431	112210	399
298	-430	122121	475
299	-429	221201	694
300	-428	212022	629
301	-427	120202	425
302	-426	202011	544
303	-425	020122	179
304	-424	201220	537
305	-423	012212	158
306	-422	122120	474
307	-421	221221	700
308	-420	212222	647
309	-419	122202	479
310	-418	222011	706
311	-417	220122	665
312	-416	201202	533
313	-415	012002	137
314	-414	120020	411
315	-413	200221	511
316	-412	002222	80
317	-411	022220	240
318	-410	222200	720
319	-409	222012	707
320	-408	220102	659
321	-407	201002	515
322	-406	010002	83
323	-405	100020	249
324	-404	000221	25
325	-403	002210	75
326	-402	022100	225
327	-401	221000	675
328	-400	210012	572
329	-399	100102	254
330	-398	001011	31
331	-397	010110	93
332	-396	101100	279
333	-395	011021	115
334	-394	110210	345
335	-393	102121	313
336	-392	021201	208
337	-391	212010	624
338	-390	120112	419
339	-389	201111	526
340	-388	011122	125

341	-387	111220	375
342	-386	112221	403
343	-385	122201	478
344	-384	222001	703
345	-383	220022	656
346	-382	200202	506
347	-381	002002	56
348	-380	020020	168
349	-379	200200	504
350	-378	002012	59
351	-377	020120	177
352	-376	201200	531
353	-375	012012	140
354	-374	120120	420
355	-373	201221	538
356	-372	012222	161
357	-371	122220	483
358	-370	222221	727
359	-369	222222	728
360	-368	222202	722
361	-367	222002	704
362	-366	220002	650
363	-365	200002	488
364	-364	000002	2
365	-363	000020	6
366	-362	000200	18
367	-361	002000	54
368	-360	020000	162
369	-359	200000	486
370	-358	000012	5
371	-357	000120	15
372	-356	001200	45
373	-355	012000	135
374	-354	120000	405
375	-353	200021	493
376	-352	000222	26
377	-351	002220	78
378	-350	022200	234
379	-349	222000	702
380	-348	220012	653
381	-347	200102	497
382	-346	001002	29
383	-345	010020	87
384	-344	100200	261
385	-343	002021	61

386	-342	020210	183
387	-341	202100	549
388	-340	021012	194
389	-339	210120	582
390	-338	101212	293
391	-337	012111	148
392	-336	121110	444
393	-335	211121	610
394	-334	111222	377
395	-333	112211	400
396	-332	122101	469
397	-331	221001	676
398	-330	210022	575
399	-329	100202	263
400	-328	002011	58
401	-327	020110	174
402	-326	201100	522
403	-325	011012	113
404	-324	110120	339
405	-323	101221	295
406	-322	012201	154
407	-321	122010	462
408	-320	220121	664
409	-319	201222	539
410	-318	012202	155
411	-317	122020	465
412	-316	220221	673
413	-315	202222	566
414	-314	022202	236
415	-313	222020	708
416	-312	220212	671
417	-311	202102	551
418	-310	021002	191
419	-309	210020	573
420	-308	100212	266
421	-307	002111	67
422	-306	021110	201
423	-305	211100	603
424	-304	111012	356
425	-303	110111	337
426	-302	101101	280
427	-301	011001	109
428	-300	110010	327
429	-299	100121	259
430	-298	001201	46

431	-297	012010	138
432	-296	120100	414
433	-295	201021	520
434	-294	010222	107
435	-293	102220	321
436	-292	022221	241
437	-291	222210	723
438	-290	222112	716
439	-289	221102	686
440	-288	211002	596
441	-287	110002	326
442	-286	100011	247
443	-285	000101	10
444	-284	001010	30
445	-283	010100	90
446	-282	101000	270
447	-281	010021	88
448	-280	100210	264
449	-279	002121	70
450	-278	021210	210
451	-277	212100	630
452	-276	121012	437
453	-275	210111	580
454	-274	101122	287
455	-273	011211	130
456	-272	112110	390
457	-271	121121	448
458	-270	211201	613
459	-269	112022	386
460	-268	120211	427
461	-267	202101	550
462	-266	021022	197
463	-265	210220	591
464	-264	102212	320
465	-263	022111	229
466	-262	221110	687
467	-261	211112	608
468	-260	111102	362
469	-259	111011	355
470	-258	110101	334
471	-257	101001	271
472	-256	010001	82
473	-255	100010	246
474	-254	000121	16
475	-253	001210	48

476	-252	012100	144
477	-251	121000	432
478	-250	210021	574
479	-249	100222	269
480	-248	002211	76
481	-247	022110	228
482	-246	221100	684
483	-245	211012	599
484	-244	110102	335
485	-243	101011	274
486	-242	010101	91
487	-241	101010	273
488	-240	010121	97
489	-239	101210	291
490	-238	012121	151
491	-237	121210	453
492	-236	212121	637
493	-235	121222	458
494	-234	212211	643
495	-233	122122	476
496	-232	221211	697
497	-231	212122	638
498	-230	121202	452
499	-229	212011	625
500	-228	120122	422
501	-227	201211	535
502	-226	012122	152
503	-225	121220	456
504	-224	212221	646
505	-223	122222	485
506	-222	222211	724
507	-221	222122	719
508	-220	221202	695
509	-219	212002	623
510	-218	120002	407
511	-217	200011	490
512	-216	000122	17
513	-215	001220	51
514	-214	012200	153
515	-213	122000	459
516	-212	220021	655
517	-211	200222	512
518	-210	002202	74
519	-209	022020	222
520	-208	220200	666

521	-207	202012	545
522	-206	020102	173
523	-205	201020	519
524	-204	010212	104
525	-203	102120	312
526	-202	021221	214
527	-201	212210	642
528	-200	122112	473
529	-199	221111	688
530	-198	211122	611
531	-197	111202	371
532	-196	112011	382
533	-195	120101	415
534	-194	201001	514
535	-193	010022	89
536	-192	100220	267
537	-191	002221	79
538	-190	022210	237
539	-189	222100	711
540	-188	221012	680
541	-187	210102	578
542	-186	101002	272
543	-185	010011	85
544	-184	100110	255
545	-183	001121	43
546	-182	011210	129
547	-181	112100	387
548	-180	121021	439
549	-179	210201	586
550	-178	102022	305
551	-177	020211	184
552	-176	202110	552
553	-175	021112	203
554	-174	211120	609
555	-173	111212	374
556	-172	112111	391
557	-171	121101	442
558	-170	211001	595
559	-169	110022	332
560	-168	100211	265
561	-167	002101	64
562	-166	021010	192
563	-165	210100	576
564	-164	101012	275
565	-163	010111	94

566	-162	101110	282
567	-161	011121	124
568	-160	111210	372
569	-159	112121	394
570	-158	121201	451
571	-157	212001	622
572	-156	120022	413
573	-155	200211	508
574	-154	002122	71
575	-153	021220	213
576	-152	212200	639
577	-151	122012	464
578	-150	220111	661
579	-149	201122	530
580	-148	011202	128
581	-147	112020	384
582	-146	120221	430
583	-145	202201	559
584	-144	022022	224
585	-143	220220	672
586	-142	202212	563
587	-141	022102	227
588	-140	221020	681
589	-139	210212	590
590	-138	102102	308
591	-137	021011	193
592	-136	210110	579
593	-135	101112	284
594	-134	011111	121
595	-133	111110	363
596	-132	111121	367
597	-131	111201	370
598	-130	112001	379
599	-129	120001	406
600	-128	200001	487
601	-127	000022	8
602	-126	000220	24
603	-125	002200	72
604	-124	022000	216
605	-123	220000	648
606	-122	200012	491
607	-121	000102	11
608	-120	001020	33
609	-119	010200	99
610	-118	102000	297

611	-117	020021	169
612	-116	200210	507
613	-115	002112	68
614	-114	021120	204
615	-113	211200	612
616	-112	112012	383
617	-111	120111	418
618	-110	201101	523
619	-109	011022	116
620	-108	110220	348
621	-107	102221	322
622	-106	022201	235
623	-105	222010	705
624	-104	220112	662
625	-103	201102	524
626	-102	011002	110
627	-101	110020	330
628	-100	100221	268
629	-99	002201	73
630	-98	022010	219
631	-97	220100	657
632	-96	201012	518
633	-95	010102	92
634	-94	101020	276
635	-93	010221	106
636	-92	102210	318
637	-91	022121	232
638	-90	221210	696
639	-89	212112	635
640	-88	121102	443
641	-87	211011	598
642	-86	110122	341
643	-85	101211	292
644	-84	012101	145
645	-83	121010	435
646	-82	210121	583
647	-81	101222	296
648	-80	012211	157
649	-79	122110	471
650	-78	221121	691
651	-77	211222	620
652	-76	112202	398
653	-75	122011	463
654	-74	220101	658
655	-73	201022	521

656	-72	010202	101
657	-71	102020	303
658	-70	020221	187
659	-69	202210	561
660	-68	022112	230
661	-67	221120	690
662	-66	211212	617
663	-65	112102	389
664	-64	121011	436
665	-63	210101	577
666	-62	101022	278
667	-61	010211	103
668	-60	102110	309
669	-59	021121	205
670	-58	211210	615
671	-57	112112	392
672	-56	121111	445
673	-55	211101	604
674	-54	111022	359
675	-53	110211	346
676	-52	102101	307
677	-51	021001	190
678	-50	210010	570
679	-49	100112	257
680	-48	001111	40
681	-47	011110	120
682	-46	111100	360
683	-45	111021	358
684	-44	110201	343
685	-43	102001	298
686	-42	020001	163
687	-41	200010	489
688	-40	000112	14
689	-39	001120	42
690	-38	011200	126
691	-37	112000	378
692	-36	120021	412
693	-35	200201	505
694	-34	002022	62
695	-33	020220	186
696	-32	202200	558
697	-31	022012	221
698	-30	220120	663
699	-29	201212	536
700	-28	012102	146

701	-27	121020	438
702	-26	210221	592
703	-25	102222	323
704	-24	022211	238
705	-23	222110	714
706	-22	221112	689
707	-21	211102	605
708	-20	111002	353
709	-19	110011	328
710	-18	100101	253
711	-17	001001	28
712	-16	010010	84
713	-15	100100	252
714	-14	001021	34
715	-13	010210	102
716	-12	102100	306
717	-11	021021	196
718	-10	210210	588
719	-9	102112	311
720	-8	021111	202
721	-7	211110	606
722	-6	111112	365
723	-5	111111	364
724	-4	111101	361
725	-3	111001	352
726	-2	110001	325
727	-1	100001	244

Додаток Ж. Код шаблону проєктування «Перетворювач» або «Конвертер» (Converter Design Pattern) мовою C#

```
// Converter Design Pattern
// Devides a program system into three separated modules: Source, Handler, and
Receiver,
// that allows to easily connect new devices and add new handling algorithms without
// risking the system functionality.

using System;

namespace Converter.General
{
    /// <summary>
    /// The Source interface defines the must-have operations for any sources
    /// connected to the program system.
    /// </summary>
    interface ISource
    {
        /// <summary>
        /// Allows to read data from the source in the format this source supports.
        /// </summary>
        void ReadFrom();
    }

    /// <summary>
    /// Concrete Sources implement reading depending on a specific source
    /// that is being connected to the system.
    /// </summary>
    class ConcreteSource : ISource
    {
        public void ReadFrom()
        {
            throw new NotImplementedException();
        }
    }

    /// <summary>
    /// The Receiver interface defines the must-have operations for any receivers
    /// connected to the program system.
    /// </summary>
    interface IReceiver
    {
        /// <summary>
        /// Allows to write processed data to a receiver.
        /// </summary>
        void WriteTo();
    }

    /// <summary>
    /// Concrete Receivers implement writing data to a receiver depending
    /// on the specific receiver that is being connected to the system.
    /// </summary>
    class ConcreteReceiver : IReceiver
    {
        public void WriteTo()
        {
            throw new NotImplementedException();
        }
```

```
    }

    /// <summary>
    /// If applicable, The Synchronizer defines how to convert the source data type
    /// into the data type the Handler can process.
    /// </summary>
    interface ISynchronizer
    {
        /// <summary>
        /// Implements data synchronization procedure.
        /// </summary>
        void SynchronizeTypes();
    }

    /// <summary>
    /// Concrete Synchronizers implement data synchronization procedure for each
    /// specific pair of data types.
    /// </summary>
    class ConcreteSynchronizer : ISynchronizer
    {
        public void SynchronizeTypes()
        {
            throw new NotImplementedException();
        }
    }

    /// <summary>
    /// The Handler defines a base class that extracts raw data from a source
    /// and passes processed data to a receiver.
    /// </summary>
    class Handler
    {
        /// <summary>
        /// Receives initial data from the source.
        /// </summary>
        /// <param name="source">The data source defined by the Client.</param>
        public void GetDataFromSource(ISource source)
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Transmits processed data to the receiver.
        /// </summary>
        /// <param name="receiver">The data receiver defined by the Client.</param>
        public void OutputConvertedData(IReceiver receiver)
        {
            throw new NotImplementedException();
        }
    }

    /// <summary>
    /// Additional Handlers implement ths specific logic required by the problem
domain.
    /// </summary>
    class AdditionalHandler : Handler
    {
        public AdditionalHandler(ISource source)
        {
            GetDataFromSource(source);
            if (IsSynchronizationRequired())
            {
```

```
                SynchronizeTypes();
            }
            HandleData();
        }

        /// <summary>
        /// Implements the problem domain logic.
        /// </summary>
        public void HandleData()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Checks if the source data format complies with the data type the Handler
works with.
        /// </summary>
        /// <returns></returns>
        private bool IsSynchronizationRequired()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Converts the source data format into the data type the Handler works with.
        /// </summary>
        private void SynchronizeTypes()
        {
            ISynchronizer synchronizer = new ConcreteSynchronizer();
            synchronizer.SynchronizeTypes();
        }
    }

    /// <summary>
    /// The Client class defines the domain logic of the program system.
    /// </summary>
    class Client
    {
        ISource source;
        IReceiver receiver;

        public void ClientLogic()
        {
            source = new ConcreteSource();
            receiver = new ConcreteReceiver();

            Handler handler = new AdditionalHandler(source);
            handler.OutputConvertedData(receiver);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Client client = new Client();
            client.ClientLogic();
        }
    }
}
```

Додаток 3. Приклад логістичного документу

SHIPPING LIST

(1) Shipper *(full name and address)*		(4) Document No.
(2) Consignee *(full name and address)*		(5) Receiving Manager
(3) Notify Party *(full name and address; if different than Consignee)*		(6) Forwarding Agent
(7) Place of Receipt	(8) Country of Receipt	(11) Routing *(key points during delivery)*
(9) Place of Delivery	(10) Country of Delivery	

(12) Container No. (13) Total Number of Containers *(in words)*	(14) Description of Package *(up to 500 symbols)*	(15) Weight *(measurement and gross weight)* (16) Declared Value *(in dollars)*
(17) Charges	(18) Prepaid	(19) Service Type
(20) Collected at		(21) Date of Collection