Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

до бакалаврського дипломного проекту

на тему:

«Захист в системах зберігання даних на основі контрольних сум CRC»

Виконав: студент 4 курсу, групи 2КІ-18б
спеціальності 123 - «Комп'ютерна інженерія»
Степанов О. Д.
Керівник к.т.н., доц. каф. ОТ
Семеренко В. П.
Рецензент к.т.н., доц. каф. МБІС
Карпінець В. В.

Допущено до захисту
д.т.н., проф. Азаров О.Д.
"23" червня 2022 р.

ВНТУ 2022

Інв. № ориг. | Підпис та дата | На зам. Інв.№ | Інв. № дубл. | Підпис та дата

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 123 - «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

обчислювальної техніки

проф. Азарову О.Д.

«08» 02 2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Степанову Олексію Дмитровичу

1. Тема проекту «Захист в системах зберігання даних на основі контрольних сум CRC», керівник роботи к.т.н., доц. каф. ОТ Семеренко В. П. затверджені наказом вищого навчального закладу від «24» березня 2022 року № 66.
2. Строк подання студентом проекту 21.06.2022 р.
3. Вихідні данні до проекту: огляд методів завадостійкого кодування за допомогою CRC, аналіз математичних основ лінійних автоматів, паралельний пошук контрольної суми CRC, на основі математичного апарату багатопотокових лінійних автоматів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області, методи знаходження контрольної суми, метод паралельного розрахунку контрольної суми CRC, програма для автоматизованого прискореного розрахунку контрольних сум CRC, тестування додатку, висновки, список літератури, додатки.

5. Перелік графічного матеріалу: блок-схема алгоритму, скріншоти фрагментів програми і результатів виконання обчислень, презентація.

6. Консультанти розділів проекту приведені в Таблиці 1

Таблиця 1 - Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Спеціальна частина	Семеренко В. П., доцент кафедри ОТ		

7. Дата видачі завдання « 24 » березня 2022 р.

8. Календарний план виконання БДП приведений в Таблиці 2.

Таблиця 2 - Календарний план

№ з/п	Назва етапів виконання комплексної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	07.03.22	
2	Пошук матеріалів про кодування інформації	08.03-31.03.22	
3	Пошук та аналіз типів завадостійкого кодування	04.04-15.04.22	
4	Аналіз математичних основ контрольних сум	18.04-22.04.22	
5	Аналіз розповсюджених методів знаходження контрольних сум CRC	25.04-.06.05.22	
6	Аналіз математичного апарату лінійних автоматів	09.05-13.05.22	
7	Аналіз автоматних методів пошуку контрольної суми	16.05-20.05.22	
8	Підготовка програмного та апаратного рішення	23.05-26.05.22	
9	Аналіз виконання роботи, висновки, додатки	27.05-31.05.22	
10	Перевірка якості виконання бакалаврського проекту та усунення недоліків	01.06 -.08.06.22	

Студент _____ Степанов О. Д.

Керівник роботи _____ Семеренко В. П.

Анотація

Степанов О. Д. Захист в системах зберігання даних на основі контрольних сум в CRC. Бакалаврська кваліфікаційна робота зі спеціальності 123 – Комп'ютерна Інженерія, Вінниця: ВНТУ, 2022. Пояснювальна записка містить 66 сторінки, 34 рисунків та 11 посилань.

В даній бакалаврській дипломній роботі була розроблена програма для розрахунку контрольної суми автоматним методом. На основі здійсненого аналізу предметної області було проаналізовано сучасні підходи до перевірки цілісності та достовірності даних та представлено метод знаходження контрольних сум за допомогою математичного апарату багатоканальних лінійних автоматів. Відповідно до поставленої задачі була розроблена схема та програма у середовищі Visual Studio, використовуючи мову програмування C# . Таким чином було отримано швидкодіючу програму, що прискорює знаходження контрольних сум у декілька разів.

Ключові слова: контрольна сума, RAID, лінійні автомати, автоматний метод, CRC.

Abstract

Stepanov O.D. Protection in data storage systems based on checksums in CRC. Bachelor degree in the specialty 123 – Computer Engineering, Vinnitsa: VNTU, 2022. The explanatory note contains 66 pages, 34 figures and 11 links.

In this bachelor's thesis, a program was developed for the rho-calculation of the checksum by the machine method. Based on the analysis of the subject area, modern approaches to checking the integrity and reliability of data were analyzed and the method of finding checksums using the mathematical apparatus of multichannel linear machine was presented. In accordance with the task, a scheme and a program were developed in the Visual Studio using the C# programming language. Thus, a fast-acting program was obtained, which speeds up the finding of checksums several times.

Keywords: checksum, RAID, linear machine, machine method, CRC.

ЗМІСТ

Вступ

У середині 60-х років минулого століття питання контролю стали дуже актуальними помилки в передачі, обробці та зберіганні повідомлень в радіотехнічних системах для різних цілей, комунікаційних технологій, космічних і військових систем, при побудові надійних цифрових пристроїв. Завадостійкі коди - це один із найефективніших засобів забезпечення високої достовірності передачі інформації. Розвиток напрямку зумовлений використанням мікропроцесорної техніки у системах зв'язку. Завадостійке кодування базується на теоремі Шеннона для передачі дискретної інформації по каналу із завадами: "Імовірністть помилкового декодування може бути як завгодно малою при виборі відповідного способу кодування". Теорема вказує на принципову можливість, але не визначає спосіб кодування. Це обумовило інтерес до розробки завадостійких кодів. Під завадостійкими розуміють коди, які дозволяють виявляти, чи виявляти і виправляти помилки, які виникли через вплив завад.

У середині 60-х років минулого століття питання контролю стали дуже **актуальними** в телекомунікаційних та комп'ютерних системах.

Завадостійкі коди - це один із найефективніших засобів забезпечення високої достовірності передачі інформації. Завадостійке кодування базується на теоремі Шеннона для передачі дискретної інформації по каналу із завадами: "Імовірність помилкового декодування може бути як завгодно малою при виборі відповідного способу кодування". Теорема вказує на принципову можливість, але не визначає спосіб кодування. Це обумовило інтерес до розробки завадостійких кодів. Під завадостійкими розуміють коди, які дозволяють виявляти, чи виявляти і виправляти помилки, які виникли через вплив завад.

Одним із ефективних методів підвищення достовірності роботи та відмовостійкості пристроїв є використання завадостійких кодів. Тоді цифровий пристрій розглядається як канал передачі інформації, в якому інформація передається не в просторі, а у часі. Ефективність використання завадостійких кодів полягає не у високій швидкості передавання даних, а у отриманні максимальної

надійності обчислювального каналу з виправленням помилок при мінімумі апаратних витрат.

Правильний вибір завадостійкого коду залежить від максимального врахування особливостей технічної діагностики сучасних цифрових пристроїв.

Зараз процес запису даних з використанням завадостійкого кодування відбувається послідовно, а це означає що дані записуються коректно, але не достатньо швидко.

Враховуючи ці вимоги, найкращим рішенням будуть циклічні коди. Вони здатні виявляти і виправляти як поодинокі, так і кратні помилки. Використання математичного апарату лінійних автоматів суттєво прискорить запис достовірних даних на носій. Але необхідно удосконалити циклічний код для коректної та швидкої роботи системи, адже традиційно запис достовірних даних на носій відбувається послідовно, а тому і повільно.

Об’єктом досліджень є процеси, що протікають в паралельних засобах передавання і зберігання даних в технічних системах.

Предметом дослідження є паралельні циклічні коди на основі математичного апарату лінійних автоматів

Метою роботи є зменшення часу запису та зчитування інформації в комп’ютерних системах зберігання даних.

Для досягнення поставленної мети необхідно реалізувати наступні **задачі**:

- проаналізувати класичні методи розрахунку контрольних сум;
- проаналізувати автоматні методи розрахунку контрольних сум;
- обрати математичний апарат лінійних автоматів;
- розробити блок-схему алгоритму розрахунку контрольної суми з обраним математичним апаратом;
- обрати програмне середовище для розробки програмного забезпечення;
- протестувати програмний засіб, шляхом моделювання роботи користувача.

Методи дослідження

Для виконання завдання були використані: теорія завадостійкого кодування, теорія інформації, теорія лінійних автоматів, цифрова схемотехніка, об’єктно-орієнтоване програмування.

Практичне значення одержаних результатів

Проаналізовані та отримані дані можна використати для вдосконалення цифрових пристроїв, прискорення запису даних на носії та забезпечення цілісності та достовірності цих даних. Запропоновано використання такого захисту даних в дискових масивах RAID.

Публікації: Семеренко В.П., Степанов О.Д. Програмна реалізація CRC на основі лінійних автоматів. Матеріали XLIX Науково-технічної конференції підрозділів ВНТУ, Вінниця, 16-18 березня 2022 р.

1 Завадостійкі коди в системах передавання та збереження даних

Методи підвищення відмовостійкості систем пам'яті за допомогою кодів, стійких до пам'яті, в останні роки були предметом книг [1], доповідей на щорічних міжнародних симпозіумах з теорії інформації (ISIT), доповідей на щорічних міжнародних симпозіумах про помилки і відмовостійкість.

Завадостійкі коди для захисту пам'яті бажано використовувати як при виготовленні мікросхем пам'яті, так і при виготовленні пристроїв пам'яті на основі мікросхем. Пропонується використовувати в якості «класичних» кодів, наприклад, Хаммінга, Ріда-Соломона (RS), Боуза-Чоудхурі-Хокінгема, каскадні коди на їх основі, коди з низькою щільністю перевірок, LDPC та нові конструкції коду, наприклад, "ранг", нелінійні коди.[2]

1.1 Призначення завадостійких кодів в системах передавання даних

При передачі сигналу через будь-який канал зв’язку можуть з’явитися помилки, які можуть призвести до зпотворення інформації, що переноситься. Існує багато методів для виправлення таких помилок, але щоб виправити ці помилки, потрібно спочатку виявити ці помилки. Для цього існують певні методи, які базуються на надлишковій передаваючій інформації, що дозволяє не лише виявляти наявність спотворення, але і в певних випадках вилучати ці спотворення.[3]

Найвідомішими з методів виявлення помилок передачі даних є:

- посимвольний контроль парності;
- поблочний контроль парності;
- обчислення контрольних сум;
- контроль на основі циклічного надлишкового коду - CRC (Cyclical Redundancy Check).

Посимвольний контроль парності, також відомий як поперечний, він базується на передачі з кожних байтом додаткового біта, приймаючого одиничне значення по парній або непарній кількості одиничних бітів в контрольованому байті. Посимвольний контроль парності є простим як у програмній, так і в апаратній реалізації, але його навряд чи можна назвати ефективним методом

виявлення помилок, адже зпотворення більш ніж одного біта вихідної послідовності різко знижує вірогідність виявлення помилок передачі даних. Цей вид контролю часто реалізується апаратно в приладах зв'язку.

Поблочний контроль парності, також відомий як повздовжній. Він базується на тому, що число символів які передаються як єдиний блок даних буде завчасно відоме для джерела та отримувача інформації. У даній схемі контролю для кожної позиції розрядів в символах блоку (поперек блоку) розраховуються свої біти парності, які додаються у вигляді звичайного символу в кінець блоку. На відміну від посимвольного контролю парності поблочний контроль парності має більші можливості для виявленню та навіть корегуванню помилок передачі, але всеодно він не може виявити певні типи помилок.

Обчислення контрольних сум. На відміну від інших методів для цього методу нема чіткого визначення алгоритму. У найпростішому вигляді контрольна сума - це арифметична сума двійкових значень блоку символів, що контролюється. Але цей і попередні методи мають спільні недоліки, серед яких найголовнішим є нечутливість контрольної суми до парної кількості помилок в одній колонці та порядку слідування символів у блоці.

Контроль на основі циклічного надлишкового коду - CRC (Cyclical Redundancy Check). Цей метод є більш потужним і широко використовуваним методом виявлення помилок передачі інформації. Він забезпечує виявлення помилок з високою вірогідністю.

Завадостійкість кодування забезпечується за рахунок внесення надлишковості в кодові комбінації, тобто крім інформаційних є і надлишкові (додаткові) розряди.

Перші спроби створення кодів з надлишковою інформацією почалися задовго до появи сучасних ПК. Як приклад, ще в шістдесятих роках минулого століття Рідом і Соломоном була розроблена ефективна методика кодування - код Ріда-Соломона. Використання її у ті часи не представлялося можливим, так як провести операцію декодування за оптимальний час першими алгоритмами не

вдавалося. Крапку в цьому питанні поставила фундаментальна робота Берлекампа, опублікована в 1968 році. Ця методика, на практичне застосування якої вказав через рік Мессі, і донині використовується в цифрових пристроях, що забезпечують приймання RS-кодованих даних. Більш того: дана система дозволяє не тільки визначати позиції, але й виправляти невірні кодові символи (найчастіше октети).

Але далеко не всюди від коду потрібна корекція помилок. На сьогодні канали зв'язку мають прийнятні характеристики, і часто достатньо лише перевірити, чи успішно пройшла передача або виникли будь-які складності; структура ж помилок і конкретні позиції невірних символів абсолютно не цікавлять сторону, яка приймає дані. І в цих умовах дуже вдалим рішенням виявилися алгоритми, що використовують контрольні суми. CRC як найкраще підходить для подібних задач: невисокі витрати ресурсів, простота реалізації і вже сформований математичний апарат з теорії лінійних циклічних кодів забезпечили їй величезну популярність.

1.2 Призначення завадостійких кодів в системах збереження даних

Вперше суть роботи RAID-масивів була описана в науковій статті в 1987 році. За майже 30 років технологія не змінилася в принципі, але її популярність зросла. Перш за все, RAID (надлишковий масив незалежних дисків) призначений для підвищення надійності безпеки даних і швидкості зчитування або запису інформації, обумовлений різним ступенем відмовостійкості і швидкості.[4]
Однак в залежності від порядкового номера масиви виконують різні функції, наприклад, класифікують їх різними цифрами від 0 до 7. Також існує декілька комбінованих варіацій як RAID 10, який поєднує розширюваність RAID 0 і надійність RAID 1. Найбільш популярними є 0, 1, 3 і 5. Якщо раніше ця технологія використовувалася тільки на дорогих серверах, на яких використовувалися диски SCSI, то зараз ситуація діаметрально протилежна, RAID-масиви використовуються на недорогих, тобто серверах початкового рівня. RAID (Redundant Array of Independent Disks) - масив дискових сховищ, конфігурація яких залежить

від швидкості, безпеки сервера і розміщених на ньому даних клієнта.[5]

Просунуті надлишкові системи створюються для компромісу між високою швидкістю доступу, місткістю сховища та відмовостійкістю. Ці системи зазвичай базуються на ідеї чергування з RAID 0, але дані розширюються додатковою інформацією - інформацією парності, яка додає надлишковість та дозволяє відновити файли або навіть продовжити роботу зі сховищем після виходу з ладу його компонента. Такі системи включають RAID 3, RAID 4, або RAID 7 (набір страйпів із виділеною парністю), RAID 5 (набір страйпів із розподіленою парністю) та RAID 6 (набір страйпів із подвійною розподіленою парністю). Термін "одинарна" парність означає, що інформація відновлюється або система функціонує після виходу з ладу одного компонента; "подвійна" парність - до двох компонентів. RAID 3 і подібні системи використовують класичний метод RAID 0, розширений за допомогою одного додаткового диска для зберігання парності. RAID 5 і RAID 6 розподіляють парність між усіма дисками, щоб прискорити процес оновлення парності для операцій запису даних. Відновлення даних із цих систем можливе в разі неушкодженого масиву або якщо один (в RAID 3, RAID 4, RAID 5, RAID 7) чи до двох (в RAID 6) компонентів було пошкоджено.[6]

RAID 3 - це відмовостійкий масив, який має один додатковий диск, на нього записуються дані, а також здійснюється паралельний ввід-вивід. RAID 3 використовує байтовий метод чергування. Перевагою RAID 3 є швидке відновлення інформації в разі збоїв сервера (рис. 1.1).

RAID 4 в деяких відносинах схожий на RAID 3, але цей рейд має великий розмір блоку записаних даних. Метод, реалізований у RAID 4, базується на звичайному наборі страйпів (як і в RAID 0), розширеному за допомогою спеціального компонента для зберігання інформації про парність для контролю помилок.

Масив має схожі з RAID 0 особливості такі, як швидкі операції читання та велика місткість, у той же час цей рівень включає власну функцію розширеної внутрішньої корекції помилок. Якщо якийсь страйп стає нечитабельним,

контролер здатний відновити його на основі інформації з інших страйпів і парності. Диск, призначений для парності, використовується не для зберігання даних, а скоріше як резервний компонент. (рис. 1.2)

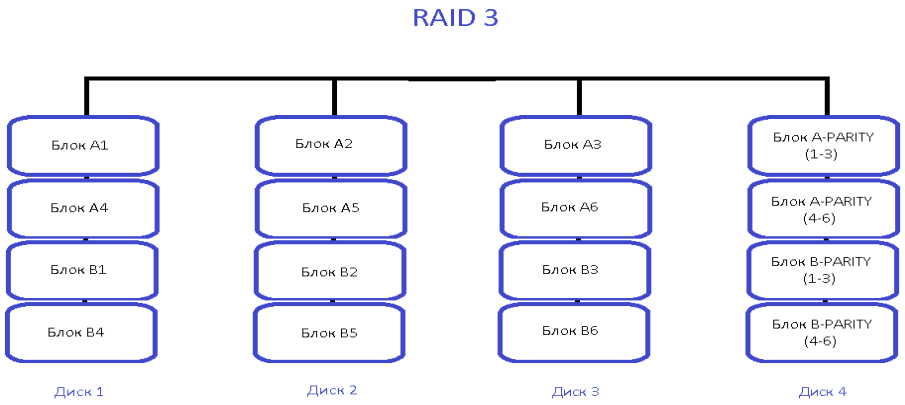


Рисунок 1.1 - Схематичне зображення структури RAID 3

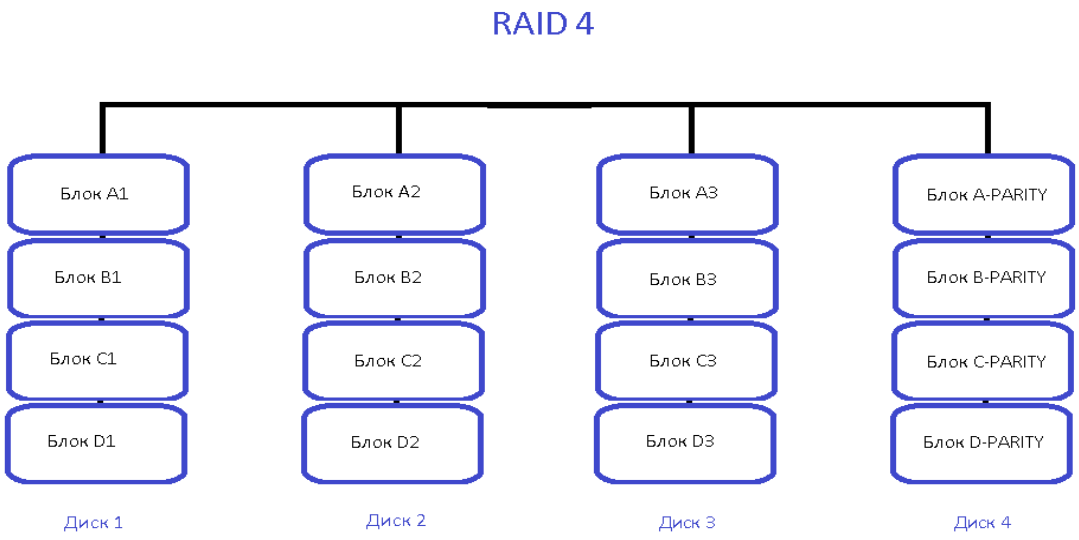


Рисунок 1.2 - Схематичне зображення структури RAID 4

Перевагами даної системи є:

- більш швидкі операції зчитування;

- висока відмовостійкість;

- продовжує працювати в режимі «обмеженої функціональності», коли один з дисків виходить з ладу.

Недоліками даної системи є:

- дуже повільні операції запису, операції вимагають оновлення інформації про парність на одному віддаленому диску ;

- повільні операції зчитування через високе навантаження на компонент з парністю ;

Найпоширеніший масив - RAID 5, для якого потрібно три і більше дисків. У ньому використовуються всі, крім одного диска, наприклад, якщо є 4 диски, то використовується 3 диски. Цей вид масиву вважається найбільш економічним. Відмовостійкість досягається тими ж засобами, що і в RAID 4: Набір страйпів зберігає фактичні дані та інформацію про парність; кожен стовпець страйпів підсумовується у страйп парності стовпця. RAID 5 поєднує в собі функції RAID рівня 0 (швидкі операції читання і велика місткість) та RAID 4 (розширену внутрішню корекцію помилок). Якщо страйп стає не читабельним, контролер може відновити його на основі інших страйпів та інформації про парність. Даний рейд використовується для великої кількості завдань, продуктивність підвищується з додаванням додаткових дисків (рис. 1.3).

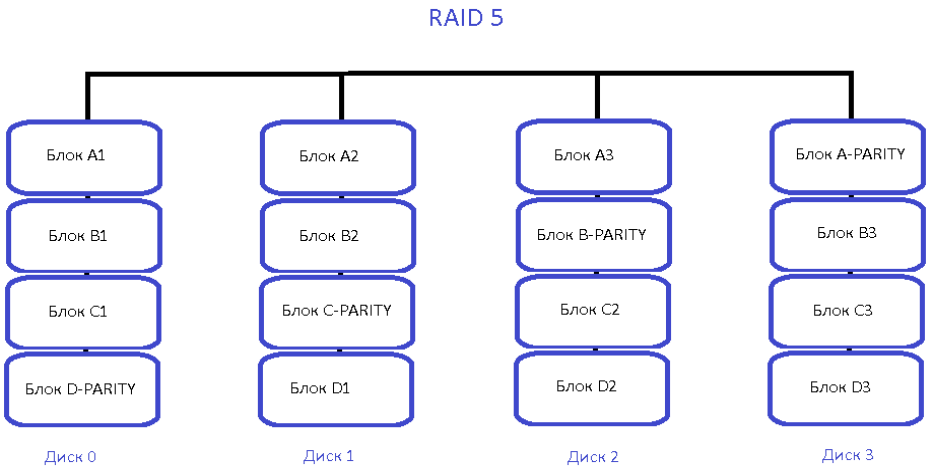


Рисунок 1.3 - Схематичне зображення структури RAID 5

Перевагами такої системи є:

- процеси зчитування даних дуже швидкі, але процеси запису трохи повільніші через парність, яку потрібно розрахувати;

- у випадку збою диску у користувача досі є доступ до усіх даних, навіть якщо несправний диск замінюється, а контролер сховища відновлює дані на новому диску;

- відмовостійкість;

Недоліками системи є:

- швидкість операції запису залежить від вмісту і методу розподілу парності;

- якщо один з дисків масиву, який має великий об’єм диску, виходить зладу і замінюється, то відновлення даних може зайняти більше доби, в залежності від навантаження на масив і швидкості контролера;

- якщо в процесі відновлення другий диск також вийде з ладу, то дані будуть втрачені назавжди.

RAID 6 було створено з метою розширення RAID рівня 5 за допомогою ще одного страйпа для надлишковості даних. Для цього застосовується алгоритм коду Ріда-Соломона на основі алгебри полів Галуа. Цей метод дозволяє додати ще один компонент для надлишковості даних та ефективно виправляти помилки. Організація даних на RAID 6 аналогічна RAID 5: дані та парність (P-страйп) розподіляються по компонентах сховища. Різниця полягає в додатковому страйпі (Q-страйпі), який розташований разом із P-страйпом та містить суму даних GF (рис. 1.4).[7]

Перевагами даної системи є:

- більш швидкі операції читання ;

- висока відмовостійкість ;

- масив може працювати в режимі обмеженої функціональності, коли один, або навіть два диски виходять з ладу.

Недоліками даної системи є:

- більш повільні операції запису в порівнянні з RAID 0;

- швидкість операцій запису залежить від вмісту і методу розподілу парності;

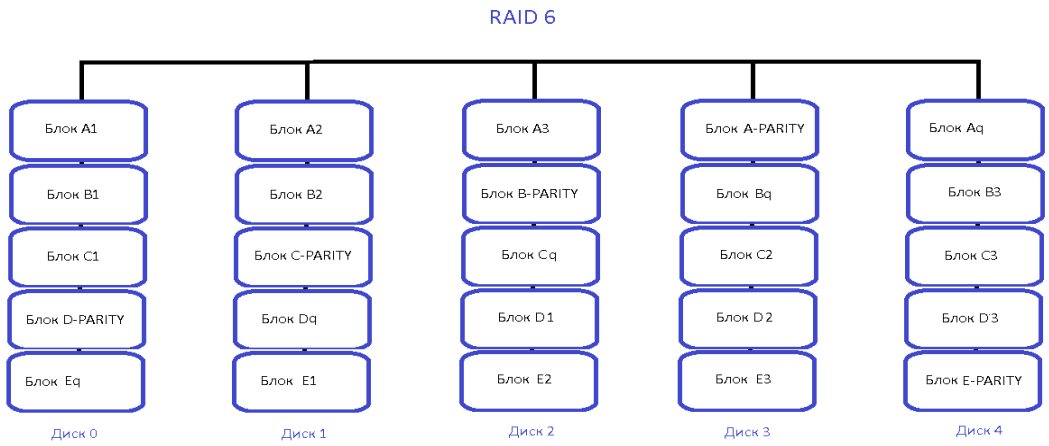


Рисунок 1.4 - Схематичне зображення структури RAID 6

RAID 7 - це рівень RAID, який включає вбудовану операційну систему та процесор у режимі реального часу для розширених операцій зчитування/запису /запису або вводу-виводу даних та можливостей кешування даних. RAID 7 в основному включає в себе функції з RAID рівня 3 і 4. RAID 7 має інтегрований кеш і спеціально побудований процесор для управління масивом, який допомагає в досягненні більш швидких операцій зчитування/запису даних. Він також має меншу залежність від парності дисків за рахунок додавання апаратного забезпечення контролера (кеша і процесора), а також забезпечує потрійну парність. Будучи власною технологією, вона вимагає спеціалізованого контролера для читання/запису даних (рис. 1.5).

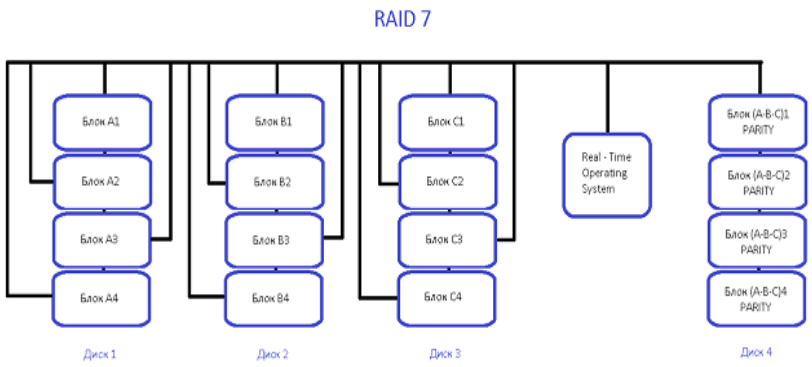


Рисунок 1.5 - Схематичне зображення структури RAID 7

RAID 10 застосовує дзеркалювання наборів страйпів для підвищення відмовостійкості без шкоди для продуктивності сховища. RAID 10 застосовує

розширення страйпів для дзеркалювання даних, покращуючи продуктивність та ефективність використання місткості сховища. Масив вимагає не менше чотирьох дисків. Відновити дані із такої системи досить просто: потрібно взяти будь-який непошкоджений компонент із кожного дзеркала та побудувати RAID 0 поверх нього у віртуальному режимі.(рис. 1.6).

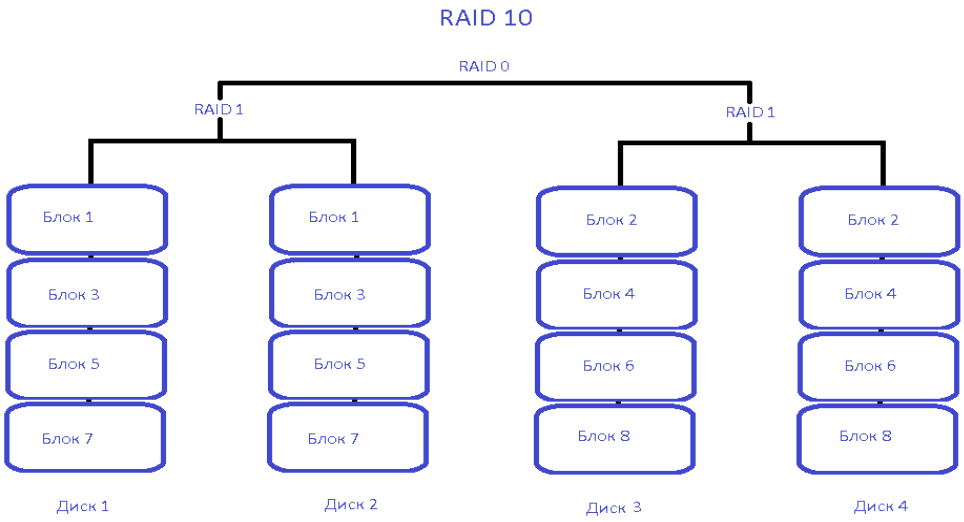


Рисунок 6 - Схематичне зображення структури RAID 10

Перевагами такої системи є:

- зчитування даних виконується дуже швидко;

- якщо два диски вийдуть з ладу, ми все одно матимемо доступ до усіх даних, навіть якщо ці диски замінюються. Тому він безпечніший, ніж RAID 5.

Недоліками системи є:

- дорогі рішення, оскільки більша частина дискового простору використовується для дзеркал;

- відновлення масива, в якому вийшов з ладу один диск може зайняти багато часу;

- складні в управлінні та підтримці.

1.3 Порівняльна характеристика КС

У найзагальнішому своєму вигляді контрольна сума являє собою деяке

значення, побудоване за певною схемою на основі кодованого повідомлення. Перевірочна інформація при систематичному кодуванні дописується, найчастіше, на кінець повідомлення- після корисних даних. З приймального боку абонент знає алгоритм обчислення контрольної суми: відповідно, програма має можливість перевірити коректність прийнятих даних.

При передачі пакетів по реальному каналу, зрозуміло, можуть виникнути спотворення вихідної інформації внаслідок різних зовнішніх впливів: електричних наведень, поганих погодних умов і багатьох інших. Сутність методики в тому, що при хороших характеристиках хеш-функції в переважній кількості випадків помилка в повідомленні призведе до зміни обчисленого на прийомі значення CRC. Якщо вихідна і обчислена суми не рівні, ухвалюється рішення про недостовірність отриманих даних, і можна запитати повторну передачу пакета.

Контрольні суми застосовуються при зберіганні даних в пам'яті (оперативної і постійної), при зберіганні даних на магнітних носіях (дисках, стрічках), в локальних і глобальних мережах передачі інформації. При використовуванні контрольної суми для захисту збереженої інформації можна встановити наявні-сть або відсутність пошкоджень даного масиву (файлу, сектору на диску). При використовуванні контрольної суми для захисту передаваної по мережі інфомації приймач може потребувати від передавача повторну передачу спотвореного масиву.

Алгоритм CRC базується на властивості ділення з остачею двійкових поліномів, тобто поліномів над скінченним полем GF(2). Значення CRC є по суті остачею від ділення многочлена, відповідного вхідним даним, на деякий фіксований породжувальний многочлен.

Кожній послідовності бітів $a_0, a_1, a_2, \dots, a_{(N-1)}$ взаємно однозначно зіставляється двійковий многочлен $\sum_{n=0}^{(N-1)}$, послідовність коефіцієнтів якого являє собою початкову послідовність. Наприклад, послідовність бітів 1011010 відповідає многочлену:

$$P(x) = 1 * x^6 + 0 * x^5 + 1 * x^4 + 1 * x^3 + 0 * x^2 + 1 *$$
$$* x^1 + 0 * x^0 = x^6 + x^4 + x^3 + x^1$$

Кількість різних многочленів степені меншої *N* дорівнює 2^N, що збігається з числом всіх двійкових послідовностей довжини N. Значення контрольної суми в алгоритмі з породжуючим многочленом G(x) степені N задається як бітова послідовність довжини N, яка представляє многочлен R(x), отриманий в остачі при діленні многочлена P(x), який представляє вхідний потік біт, на многочлен

$$G(x): R(x) = P(x) \times x^N \; mod \; G(x),$$

де R(x) - многочлен, який представляє значення CRC;

P(x) - многочлен, коефіцієнти якого представляють вхідні дані;

G(x) - породжувальний многочлен;

N - степінь породжувального многочлена.

Множення x^N здійснюється приписуванням нульових бітів до вхідної послідовності, що покращує якість гешування для коротких вхідних послідовностей. При діленні з остачею початкового многочлена на породжуючий многочлен G(x) степені N можна отримати 2^N різних остач від ділення. G(x) найчастіше є незвідним многочленом. Зазвичай його підбирають відповідно до вимог до геш-функції у контексті кожного конкретного застосування. Проте, існує багато стандартизованих породжувальних многочленів, що володіють хорошими математичними та кореляційними властивостями (мінімальне число колізій, простота обчислення).[8]

Існує безліч способів обчислення контрольної суми, що розрізняються ступенем складності обчислення і надійністю виявлення помилок. Але найбільше поширення набув в даний час так званий «циклічний метод контролю по надлишковості» або CRC, при якому застосовується циклічна контрольна сума.

То ж виділимо декілька широко використовуваних методів знаходження CRC:

- метод ділення поліномів;

- табличний метод.

У методі ділення поліномів вихідна послідовність бітів, якою може бути навіть великий файл представляється єдиною послідовністю бітів. Ця послідовність ділиться на деяке фіксоване двійкове число (поліном). У даному методі нас цікавить остача від ділення, яка і буде значенням CRC. Ми будемо розглядати арифметику по модулю 2, коли коефіцієнти додаються без переносу, тобто коефіцієнти можуть приймати значення лише 0 або 1. Нехай послідовність біт має наступний вигляд: 110100101 . Як поліном візьмемо число 1100. Розрядні-сть поліному береться на одиницю більша, ніж необхідна розрядність контрольної суми (остачі від розподілу). Так, щоб одержати 8-розрядну остачу (8-розрядну контрольну суму), треба брати 9-розрядний поліном. В нашому випадку поліном 5-розрядний, отже, остачі буде 4-розрядною. Для отримання 8-розрядної остачі можна використовувати, наприклад, поліном 100011101 (рис. 1.7). Розподіл по модулю 2 проводиться точно так, як і звичний розподіл «в стовпчик», але замість віднімання в даному випадку використовується порозрядне додавання по модулю 2, тобто кожний результуючий біт є функцією що Виключного АБО від відповідних бітів складових. Частка від розподілу нас не цікавить, а остача, рівна в нашому прикладі 1101, і буде циклічною контрольною сумою. Але цей метод виконується досить повільно.

У табличному методі числа є остачею від розподілу по модулю 2 числа з n кінцевими нулями і з n початковими розрядами, рівними номеру числа в таблиці(Таблиця 3). Розподіл проводиться на вибраний поліном (в нашому випадку 9-розрядний). Таблиця обчислюється один раз і зберігається на диску або в ПЗП. Алгоритм обчислення контрольної суми за допомогою цієї таблиці наступний (розглядаємо випадок n = 8). Беремо перший байт нашого інформаційного масиву. Розглядаємо його як адресу в таблиці. Беремо з таблиці число з одержаним номером - одержуємо остачу O_1. Беремо другий байт масиву і складаємо його по модулю 2 із остачею O_1 Одержане число використовуємо як адресу в таблиці. За

цією адресою вибираємо з таблиці остачу O_2. Беремо третій байт масиву, складаємо його по модулю 2 із остачею O_2. Використовуючи це число як адресу в таблиці, вибираємо з неї остачу Оз і так продовжуємо до останнього байта масиву. Природно, це буде набагато швидше, ніж обчислення «в стовпчик».[9]

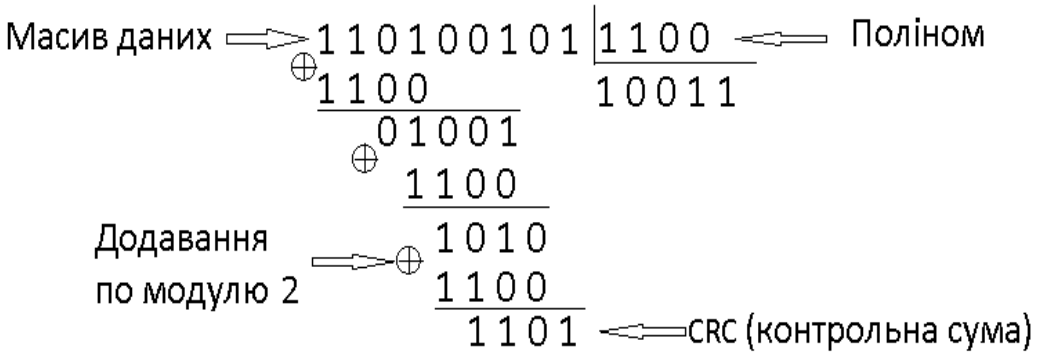


Рисунок 1.7 - Метод ділення на поліном

Таблиця 3 - Табличний метод

Адреса в таблиці	Дані в таблиці
0	0
1	Остача від ділення числа 1 0000 0000 на поліном
2	Остача від ділення числа 10 0000 0000 на поліном
3	Остача від ділення числа 11 0000 0000 на поліном
4	Остача від ділення числа 100 0000 0000 на поліном
…	Остача від ділення числа … 0000 0000 на поліном
255	Остача від ділення числа 1111 1111 0000 0000 на поліном

2 Методи обчислення контрольних сум на основі теорії лінійних автоматів

2.1Математичні основи лінійних автоматів

Розглянемо лінійний фільтр, що складається із r елементів пам'яті. Кожен такий елемент може перебувати в одному з станів q, відповідних величинам поля GF(q). Нехай s_i - стан і-ої комірки пам'яті. Позначимо через $s = (s_{r-1},..., s_0)$, $s_i \in GF(q)$, вектор, що показує стан фільтру. Нехай $s = (s_{r-1},..., s_0)$ і $s' = (s'_{r-1},\ldots, s'_0)$ - пара суміжних у часі станів, s - стан в даний момент і s' - стан в наступний за даним моментом часу. Фільтр називається лінійним, оскільки в загальному випадку його стан описується системою лінійних рівнянь над полем GF(q) :

$$\begin{cases} s'_{r-1} = c_{r-1}s_{r-1} + \ldots + c_{r-1,0}s_0 + \alpha b_{r-1}, \\ s'_{r-2} = c_{r-2}s_{r-1} + \ldots + c_{r-2,0}s_0 + \alpha b_{r-2}, \\ s'_0 = c_{0,r-1}s_{r-1} + \ldots + c_{r-1,0}s_0 + \alpha b_0, \end{cases} \quad (2.1)$$

де α - це значення символу на вході фільтра в даний момент часу.

В матричному представленні ці рівняння мають такий вигляд:

$$s' = sC + \alpha b, \quad (2.2)$$

$$C = \begin{bmatrix} c_{r-1,r-1} & c_{r-1,r-2} & \ldots & c_{r-1,0} \\ c_{r-2,r-1} & c_{r-2,r-2} & \ldots & c_{r-2,0} \\ \ldots & \ldots & \ldots & \ldots \\ c_{0,r-1} & c_{0,r-2} & \ldots & c_{0,0} \end{bmatrix}, b = [\, b_{r-1}, b_{r-2}, \ldots, b_0 \,] \quad (2.3)$$

Матриця C називається перехідною матрицею фільтру, а $c_{i,j}$ є коефіцієнтом, з яким даний стан і-ої комірки входить в суму, що визначає наступний стан j-ої комірки, вектор $b = (\, b_{r-1}, b_{r-2}, \ldots, b_0 \,)$ описує вхідні ланцюжки фільтру [10].

Наприклад, ми маємо двійковий лінійний фільтр з трьома комірками пам'яті, $r = 3$ (рис. 2.1). Він описується наступною системою рівнянь, які пов'язують наступний стан $s' = (\, s'_2, s'_1, s'_0 \,)$ з попереднім станом

$s = (s_2, s_1, s_0)$ і з входом фільтру, у цьому випадку усі змінні приймають два значення 0 і 1, і операції над ними виконуються по модулю 2 :

$$\begin{cases} s_2' = s\,1\,, \\ s_1' = s_0 + s_2 + \alpha, \\ s_0' = s_2 + \alpha, \end{cases} \tag{2.4}$$

або в матричному представленні:

$$s' = sC + \alpha b\,, \text{де } C = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, b = [0\ 1\ 1]$$

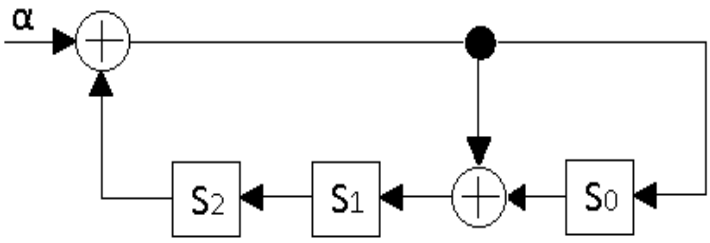


Рисунок 2.1 - Приклад лінійного фільтру.

В автономному режимі (тобто коли вхідні сигнали відсутні) фільтр залишається в нульвому стані, якщо початковий стан нульовий, а якщо початковий стан не нульовий, то фільтр пробігає деяку періодичну послідовність станів. Наприклад, якщо початковий стан був (0 0 1), то фільтр буде послідовно приймати такі стани: (0 0 1), (0 1 0), (1 0 0), (0 1 1), (1 1 0), (1 1 1), (1 0 1), (0 0 1)… і так далі з періодом 7.

Перейдемо від символьної форми С до поліноміальної форми С(х) перехідної матриці. Зробимо це таким чином:

$$C(x) = C \times \begin{bmatrix} x^{r-1} \\ x^{r-2} \\ \dots \\ 1 \end{bmatrix} = \begin{bmatrix} c_{r-1}(x) \\ c_{r-2}(x) \\ \dots \\ c_0(x) \end{bmatrix}$$

де $c_i(x)$ - поліном, що відповідає і-ому рядку матриці С.

Розглянемо поліном станів s(x) і поліном вхідних ланцюжків b(x):

$$s(x) = \sum_{i=0}^{r-1} s_i x^i \, , \, s'(x) = \sum_{i=0}^{r-1} s'_i x^i \text{ і } b(x) = \sum_{i=0}^{r-1} b_i x^i \qquad (2.5)$$

В результаті маємо:

$$s'(x) = (s \cdot C + \alpha b) \cdot \begin{bmatrix} x^{r-1} \\ x^{r-2} \\ \dots \\ 1 \end{bmatrix} = s \cdot C(x) + \alpha b(x) \qquad (2.6)$$

Так ми отримали співвідношення, що встановлює зв'язок між парою суміжних у часі станів s і s' , що записані у векторній та поліноміальній формі. Для того щоб обчислювати остачі від ділення ми маємо додати вимоги до перехідної матриці С лінійного фільтра.

Припустимо, що поліноми $\{c_j(x)\}$ задовільняють наступним умовам :

$$c_j(x) = p(x)x^j mod\ g(x) \, , j = 0,1, \dots, r-1. \qquad (2.7)$$

де p(x), g(x) - довільні поліноми;

r - степінь поліному g(x).

Тоді,

$$s'(x) = p(x)s(x) + \alpha b(x)\ mod\ g(x) \, . \qquad (2.8)$$

Доведемо:

$$s \cdot C(x) = (s_{r-1}, \dots, s_0) \cdot \begin{bmatrix} x^{r-1}p(x) \\ \dots \\ p(x) \end{bmatrix} = \sum_{j=0}^{r-1} s_j x^j p(x) = s(x)p(x)\ mod\ g(x).$$

(2.9)

Нехай поліноміальна форма С(х) перехідної матриці фільтра задовільняє вище вказаним умовам і задається поліномами:

$$c_j(x) = p(x)x_j \, mod \, g(x), j = 0,1, \dots, r-1,$$

де p(x) і g(x) - довільні поліноми над полем GF(q) і $deg(g(x)) = r$.

Тоді при передачі на вхід α, суміжні у часі стани фільтру s(x) і s′(x), будуть пов'язані спів-відношенням: $s'(x) = p(x)s(x) + \alpha b(x) \, mod \, g(x)$. Якщо задати $p(x) = x$ і подати на вхід лінійного автомата з нульовим початковим станом послідовність $(a_{m-1}, a_{m-2}, \dots, a_0)$, то на m-ому такті стан буде рівним $b(x)a(x) \, mod \, g(x)$, де $a(x) = a_{m-1} \, x^{m-1} + \dots + a_0$. У цьому випадку, якщо обрати ланцюжки так, щоб $b(x) = 1$, то на m-ому такті фільтр розрахує остачу від ділення $a(x)$ на $g(x)$ [11].

Доведемо це так: оскільки початковий стан фільтра нульовий, після першого такту стан фільтра дорівнюватиме $a_{m-1}b(x)$, після другого дорівнюватиме $am-1b(x)x + am-2b(x) \times mod \, g(x)$, і так далі. Після m-ого такту стан фільтра буде:

$$a_{m-1}b(x)x^{t(m-1)} + a_{m-2}b(x)x^{t(m-2)} + \dots + a_0b(x) = b(x)a(x^t) mod \, g(x) .$$

2.2 Послідовний автоматний метод обчислення

Ефективним методом представлення CRC є автоматні моделі, які базуються на спеціальному класі скінченних автоматів в полях Галуа - лінійні автомати, або лінійні послідовнісні схеми (ЛПС).

З моменту створення циклічних кодів основним способом їх опису залишається поліноміальний спосіб. В такому випадку n-розрядній кодовій послідовності циклічного (n, k)-коду відповідатиме кодовий поліном степені (n-1) з коефіцієнтами з поля Галуа.

$$z(x) = z_0 + z_1x + z_2x^2 + \dots + z_{n-1}x^{n-1}, \; GF(q)$$

Лінійний автомат з l входами, m виходами і r елементами пам'яті у дискретні моменти часу t над полем Галуа GF(2) описуються такою функцією

переходів:

$$S(t+1) = A \times S(t) + B \times U(t), \quad (2.9)$$

а також функцією виходів:

$$Y(t) = C \times S(t) + D \times U(t), \quad (2.10)$$

де $A = |a_{ij}|_{r\times r}, B = |b_{ij}|_{r\times l}, C = |c_{ij}|_{m\times r}$, $D = |d_{ij}|_{m\times l}$ - матриці що описують лінійну послідовнісну схему;

$S(t) = |s_i|_r$ - слово стану;

$U(t) = |u_i|_l$ - вхідна послідовність;

$Y(t) = |y_i|_m$ - вихідна послідовність.

Розмірності матриць лінійних послідовнісних схем і параметри циклічного (n, k)-коду Ω пов'язані між собою коефіцієнтом r, який дорівнює числу контрольних розрядів кодового слова Z при систематичному кодуванні ($r = n - k$).

Над полем Галуа GF(2) в лінійному автоматі з одним входом і виходом можна використати такі матриці:

$$A_2 = \begin{vmatrix} 0 & 0 & 0 & \dots & g_0 \\ 1 & 0 & 0 & \dots & g_1 \\ 0 & 1 & 0 & \dots & g_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & g_{r-1} \end{vmatrix}, B = \begin{vmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \end{vmatrix}, C = [0 \quad \dots \quad 0 \quad 1], D = |0|.$$

Останній стовпчик матриці A - коефіцієнти породжувального поліному коду Ω.

Для прикладу візьмемо породжувальний поліном $g(x) = 1 + x + x^3$. Кодове слово $Z = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1]$. Автоматний метод декодування кодового слова відповідає розрахунку CRC.

Заданому поліному $g(x)$ відповідають такі матриці ЛПС:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad C = [0 \quad 0 \quad 1], \quad D = [0].$$

Операціям ділення поліномів у поліноміальній арифметиці відповідають операції рекурсивного розрахунку станів ЛПС по формулі (2.9).

Для розрахунку станів ЛПС потрібно сформувати компоненти вхідної послідовності, враховуючи що в канал зв'язку кодове слово надходить починаючи зі старших розрядів.

Тоді рівняння станів автомата для даної вхідної послідовності будуть виглядати так:

$$S(1) = A \times S(0) + B \times U(0) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [1] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$S(2) = A \times S(1) + B \times U(1) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [1] = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

$$S(3) = A \times S(2) + B \times U(2) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [0] = \begin{bmatrix} 0+0 \\ 1+0 \\ 0+1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix},$$

$$S(4) = A \times S(3) + B \times U(3) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [0] = \begin{bmatrix} 0+1 \\ 0+1 \\ 1+0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

$$S(5) = A \times S(4) + B \times U(4) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [0] = \begin{bmatrix} 0+0+1 \\ 1+0+1 \\ 0+1+0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

$$S(6) = A \times S(5) + B \times U(5) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [1] = \begin{bmatrix} 0+1 \\ 1+1 \\ 0+0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$S(7) = A \times S(6) + B \times U(6) =$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times [0] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Початковий стан $S(0)$ дорівнює нулю, а усі наступні визначаються згідно з (2.9), останній стан $S(7)$ і буде CRC сумою.

Наведений приклад продемонстрував лише основну концепцію розрахунку CRC основі теорії ЛПС. Програмно апаратна реалізація даного методу виконуються простіше і швидше. Нам не потрібно створювати характеристичні матриці ЛПС самим, достатньо мати отримане з поліному g(x) породжувальне слово, яке є множниками розрядів поліному, для матриці А воно буде визначати розмірність і останній стопчик, у даному випадку [1 1 0].

Операції множення матриць у полі Галуа також можна заманити на операції зсуву та додавання.

CRC сума попереднього прикладу на основі операцій зсуву та додавання показано на (рис. 2.2.) Матриці станів ЛПС записуються у вигляді r-розрядних стрічок. В наступну стрічку записуються зсунуті на одну позицію вправо розряди попередньої стрічки, а на місце крайнього лівого розряду записується (n-i-1)-ий розряд кодового слова Z.

Якщо останній розряд і-ої стрічки містив 0, то при зсуві вправо він втрачається, а якщо він містив 1, то виконується операція корекції, тобто наступна стрічка додається по модулю 2 до стрічки коефіцієнтів породжувального поліному.

Такий спосіб використовується при порозрядному надходженні кодової послідовності Z. Як і при використанні поліноміальної арифметики, кількість ітерацій розрахунку CRC дорівнює кількості розрядів вхідної послідовності Z.

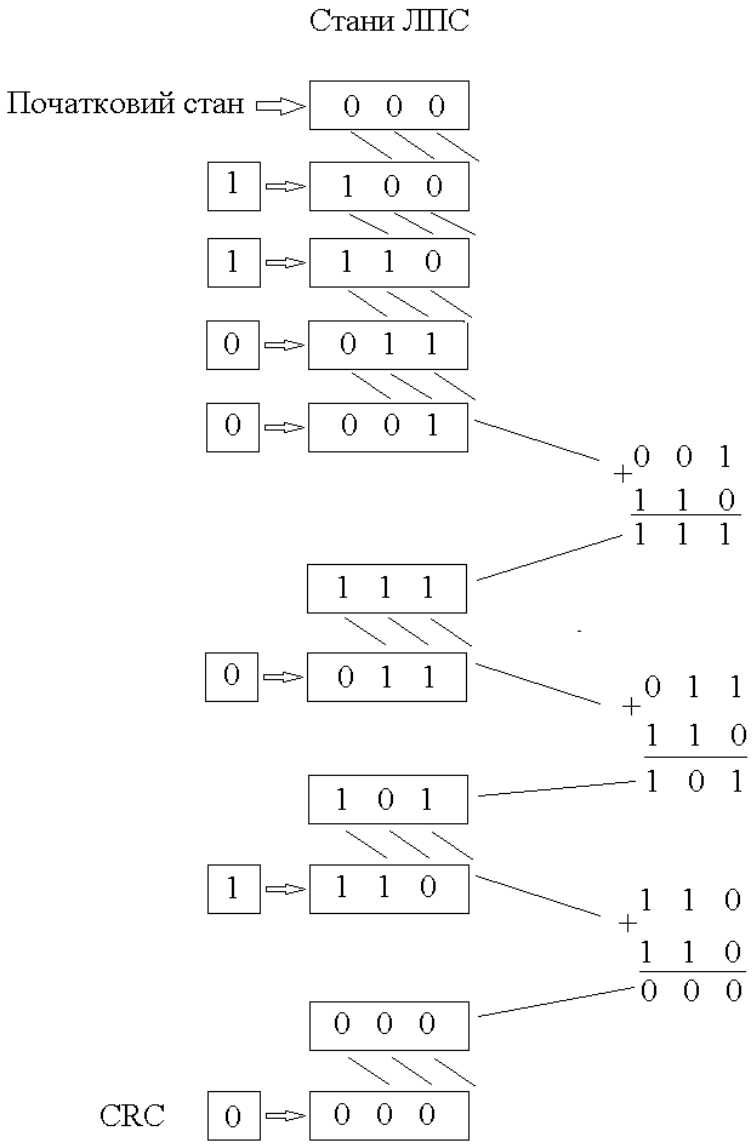


Рисунок 2.2 – Операція зсуву та додавання

2.3 Паралельний автоматний метод обчислення

Актуальною задачею, пов’язаною з CRC кодами, є прискорення розрахунку контрольних сум. Ця задача має свої особливості при програмній та апаратній реалізації, а також залежить від способу передачі даних.

Від своєї появи, CRC коди були орієнтовані на послідовну передачу даних, розрахунок контрольної суми відбувався покроково, після надходження кожного вхідного сигналу.

У зв’язку з широким використанням багатоканального зв’язку інформація

почала надходити паралельно: байтами і двійковими словами по кілька байт. Тому щоб не затримувати передачу, машина повинна розраховувати CRC набагато швидше [12].

На практиці, особливо при програмному розрахунку CRC, це дуже тяжко реалізувати, тому почали розробляти різні методи прискорення розрахунку контрольних сум. Але вони потребують або великого об'єму пам'яті для зберігання проміжних даних, або занадто складних обчислень.

Одною із найоптимальніших математичних моделей для представлення швидкісного декодування циклічного коду, що має породжувальний поліном степені p можна вважати p-канальний аналог звичайного лінійного автомату, що має p-входів, p-виходів, і такі характеристичні матриці A^p і B^p:

$$A_p = [A^p];\ \ B_p = [A^{p-1}B \quad \dots \quad AB \quad B]$$

Якщо взяти за зразок r-мірну ЛПС з минулого розділу, то її p-канальний аналог буде мати такі характеристичні матриці:

$$A_p = [A^p];\ \ B_p = \begin{bmatrix} 0 & \dots & 0 & 1 \\ 0 & \dots & 1 & 0 \\ 0 & \dots & \dots & \dots \\ 1 & \dots & 0 & 0 \end{bmatrix}. \qquad (2.11)$$

Нехай ми маємо n_w-розрядне кодове слово Z, яке складається з p-бітових символів. Тоді код стану $S(n_w)$, в який перейде p-канальна ЛПС після подачі на її входи послідовності Z по рекурсивній формулі і буде CRC сумою:

$$S(j+1) = A^r \times S(j) + z_j\,, GF(2), z \in Z, j = 1 \div n_w\,. \qquad (2.12)$$

Також цю ЛПС можна використовувати для однобітних символів кодового слова Z, це прискорить їх розрахунок. Тоді n_w-розрядне слово Z треба розділити на ϑ підслів довжини p - $Z1, \dots, Z_\vartheta$ При діленні округлення відбувається у більшу сторону і у випадку необхідності крайнє підслово доповнюється нулями, далі кожне підслово Z_i паралельно подається на p входів ЛПС.

Для порівняння оберемо ідентичний поліном $g(x) = 1 + x + x^3$ і розрахуємо CRC суму, використовуючи трьох канальну ЛПС для кодового слова

$$Z = [0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1].$$

Потрібно спочатку підготувати матрицю A^3 і матрицю B_3:

$$A^3 = \mathrm{A} \times \mathrm{A} \times \mathrm{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix};$$

$$B_3 = [A^2B \quad AB \quad B] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Тепер потрібно доповнити кодове слово Z нулями справа для того, щоб поділити його на 3 підслова:

$$Z_1 = [1 \quad 0 \quad 0]; Z_2 = [0 \quad 0 \quad 1];\ Z_3 = [0 \quad 1 \quad 0].$$

Визначимо стани трьох канальної ЛПС:

$$S''(1) = \mathrm{A}^3 \times S''(0) + B \times Z_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$S''(2) = \mathrm{A}^3 \times S''(1) + B \times Z_2 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$S''(3) = \mathrm{A}^3 \times S''(0) + B \times Z_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

В результаті ми отримаємо такий же результат, що і при послідовному розрахунку CRC суми, але у p разів швидше.

Отже, за допомогою цього методу можна паралельно розраховувати CRC при будь-якому значенні степені породжувального поліному коду (p=16,32,64),

але на відміну від матриці A послідовного методу, у матриці A^p складніша структура і операцію множення на стани неможливо замінити на операцію здвигу.

3 Програмно-апаратна реалізація методів обчислення КС

3.1 Розробка пристроїв для автоматних методів обчислення КС

Для апаратної реалізації паралельного методу обчислення контрольної суми CRC слід обрати генератор ЛПС, що формується на основі регістру зсуву з лінійними оберненими зв'язками, LFSR (Linear Feedback Shift Register).

Регістр зсуву з лінійними оберненими зв'язками має ряд переваг від-носно інших:

- прості у апаратній та програмній реалізації;
- мають високу швидкодію;
- хороші статистичні властивості формованих послідовностей.

Існує два основних типи генераторів ЛПС:

- генератор Галуа;
- генератор Фібоначі.

Схема оберненого зв'язку генератора Галуа являє собою набір операцій XOR відводних бітів з виходом генератора, що має обернений порядок бітів у регістрі: вхідний біт q_1, а вихідний q_0. (рис. 3.1)

Результат додавання по модулю 2 записується в наступну комірку, а новий біт записується у q_1. Тоді, якщо згенерований біт дорівнює нулю, то усі біти в комірках зсовуються вправо без змін, а якщо згенерований біт дорівнює одиниці, то біти відводу змінюють значення на протилежне, і всі біти зсовуються вправо.

Даний генератор не має більшу криптостійкість, ніж генератор Фібоначі, але він виграє у швидкодії, тому що усі операції XOR при паралельному використанні можна виконати за одну дію, а не послідовно.

Схема оберненого зв'язку генератора Фібоначі являє собою набір операцій XOR відводних бітів з виходом ренератора, де вхідним є біт q_0, а вихідним є біт q_1. Якщо у характеристичному поліномі $x^0 = 1$, то q_0 – вхідний біт.

Функція оберненого зв'язку має вигляд:

$$S_j = S_{j-1} \oplus S_{j-3}$$

Це означає, що наприклад для полінома $x^3 + x + 1$, бітами відводу будуть нульвий і другий біт.

При заданому поліномі $g(x) = 1 + x + x^4 + x^6 + x^7 + x^8$ при використанні генератора Фібоначі матимемо таку будову схеми (рис. 3.3).

Для апаратної реалізації паралельного методу обчислення контрольної суми CRC запропоновано використати генератор зсуву з лінійними оберненими зв'язками типу Фібоначі на основі D-тригерів і суматорів по модулю 2.

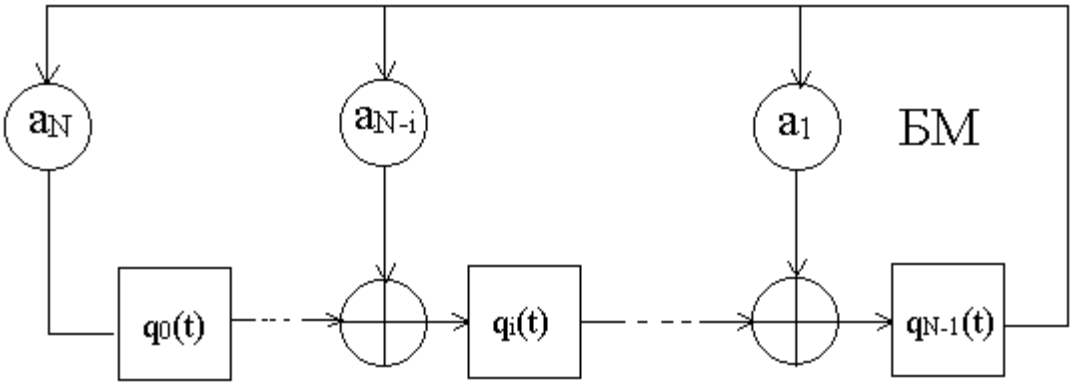


Рисунок 3.1 - генератор Галуа загального виду

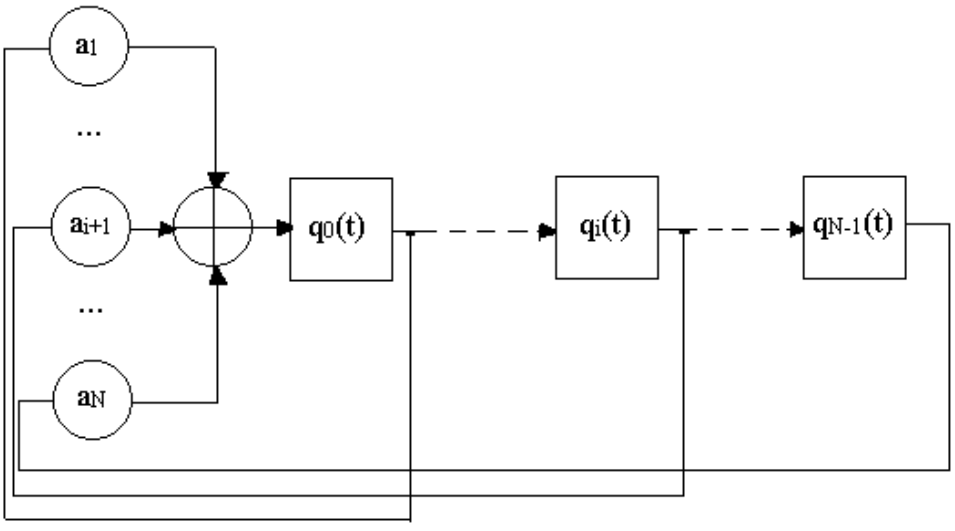


Рисунок 3.2 - генератор Фібоначі загального виду

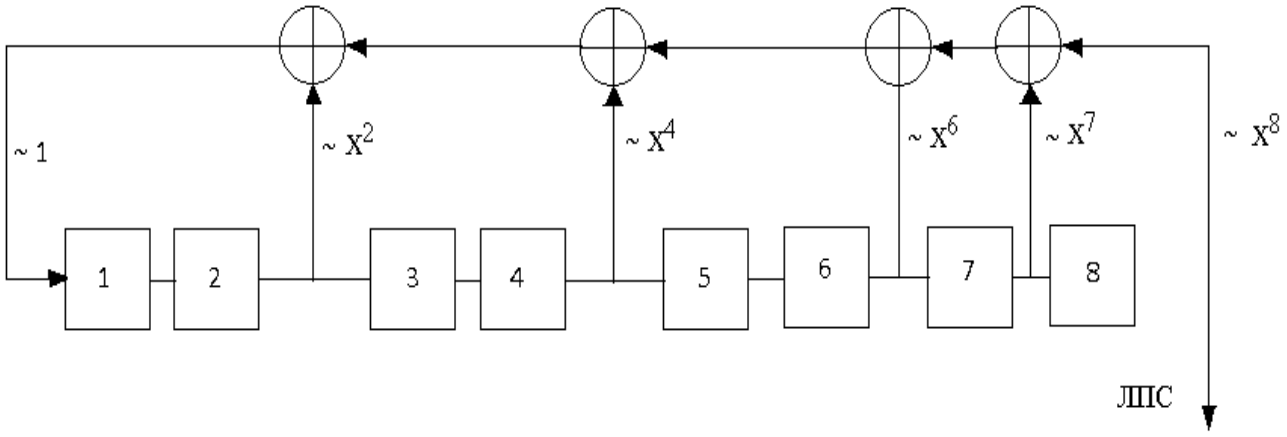


Рисунок 3.3 - генератор Фібоначі для вказаного поліному

3.2 Програмна реалізація методів обчислення КС

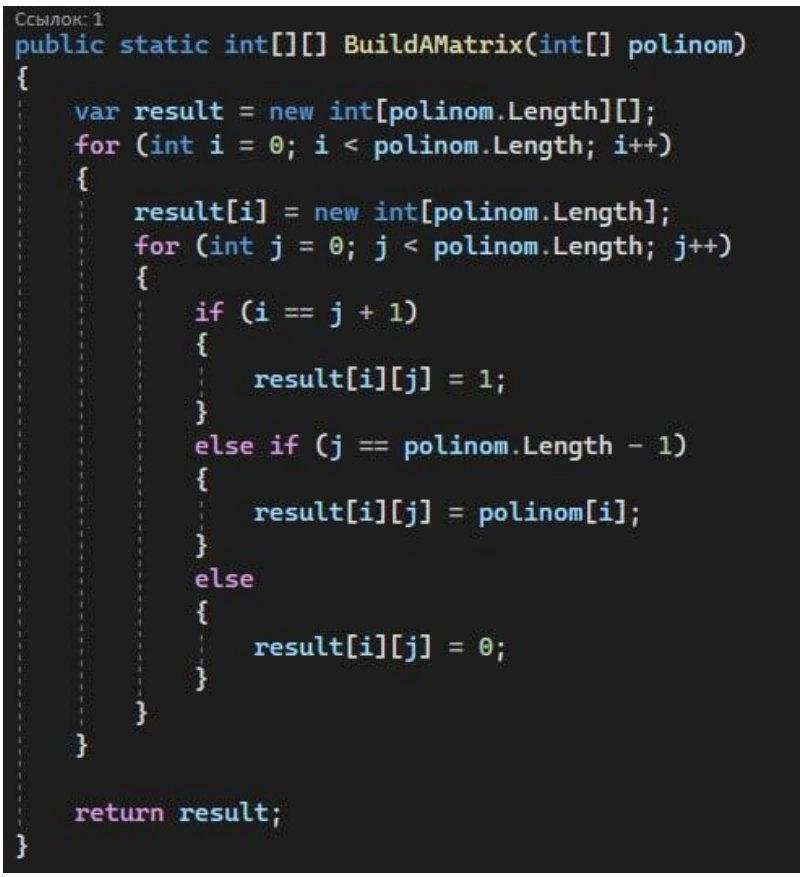
Для виконання поставленної задачі, було використано середовище програмування Visual Studio та мову програмування C#.

У програмі розраховується CRC сума для кодового слова Z. Вхідними даними для коректного виконання алгоритму програми є породжувальний поліном та кодова послідовність Z.

Функція BuildAMatrix() будує характеристичну матрицю A, використовуючи показники коефіцієнтів породжувального поліному, як дані для внесення у останній стовпчик двомірного масиву, заповнює побічну діагональ матриці одиницями, інші комірки заповнює нулями, а також використовує число кількісті цих коефіцієнтів для визначення розмірності матриці A (рис. 3.4).

Функція BuildBMatrix() будує характеристичну матрицю B, що являє собою матрицю, основна діагональ якої заповнена одиницями, усі інші комірки заповнені нулями, розмірність матриці визначається числом коефіцієнтів породжувального поліному (рис. 3.5).

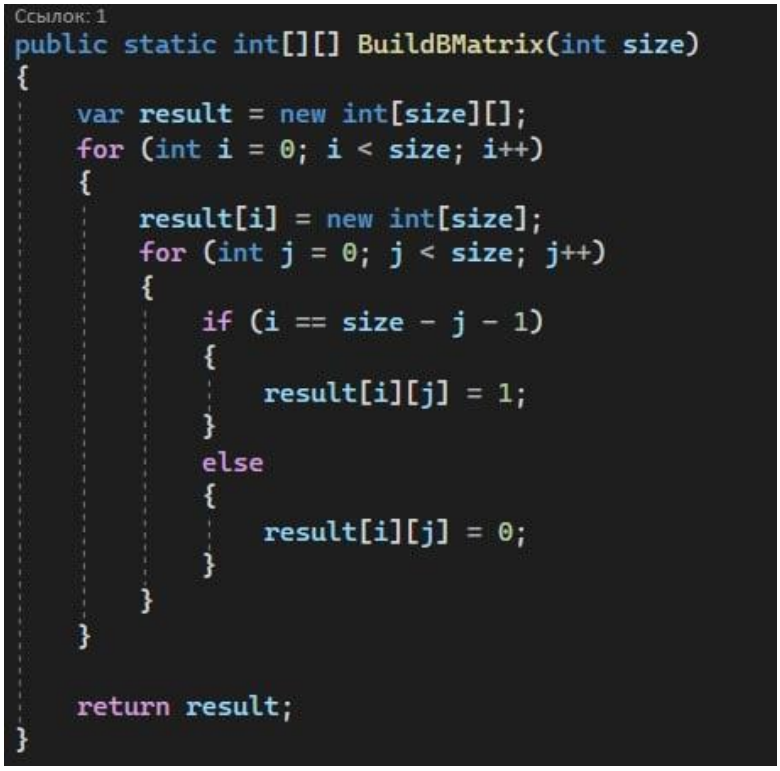
Функція PrintMatrix() виводить на екран вказану матрицю на основі циклу послідовного виведення комірок, використовуючи число коефіцієнтів породжувального поліному (рис. 3.6).



```
Ссылок: 1
public static int[][] BuildAMatrix(int[] polinom)
{
    var result = new int[polinom.Length][];
    for (int i = 0; i < polinom.Length; i++)
    {
        result[i] = new int[polinom.Length];
        for (int j = 0; j < polinom.Length; j++)
        {
            if (i == j + 1)
            {
                result[i][j] = 1;
            }
            else if (j == polinom.Length - 1)
            {
                result[i][j] = polinom[i];
            }
            else
            {
                result[i][j] = 0;
            }
        }
    }

    return result;
}
```

Рисунок 3.4 - Структура функції BuildAMatrix()



```
Ссылок: 1
public static int[][] BuildBMatrix(int size)
{
    var result = new int[size][];
    for (int i = 0; i < size; i++)
    {
        result[i] = new int[size];
        for (int j = 0; j < size; j++)
        {
            if (i == size - j - 1)
            {
                result[i][j] = 1;
            }
            else
            {
                result[i][j] = 0;
            }
        }
    }

    return result;
}
```

Рисунок 3.5 - Структура функції BuildBMatrix()

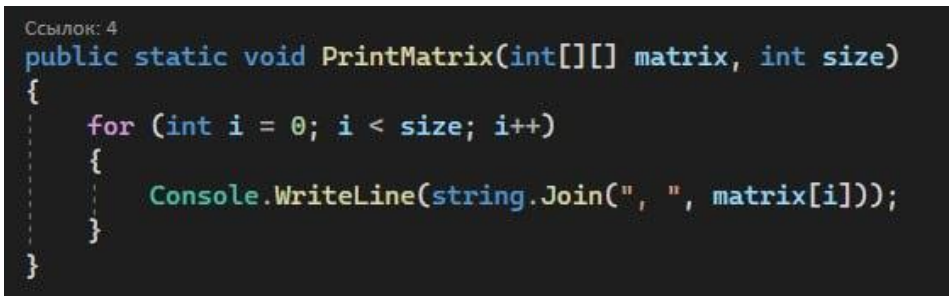


Рисунок 3.6 - Структура функції PrintMatrix()

Функція ParseZString() повертає результат перезапису вхідної послідовності Z у матрицю за допомогою циклу, заносячи вміст вхідної послідовності з кінця для зручного оперування, розмірність матриці визначається за рахунок числа коефіцієнтів породжувального поліному(рис. 3.7).

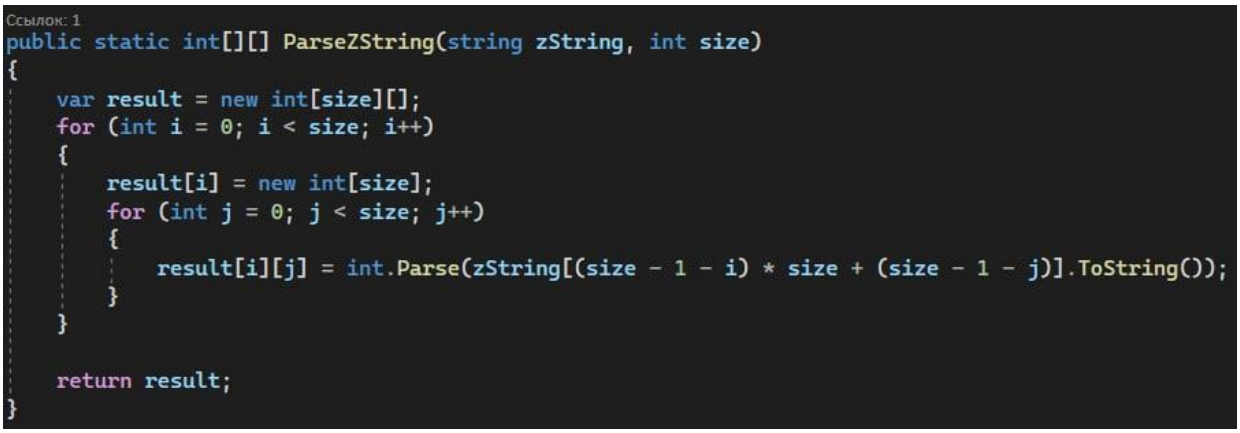


Рисунок 3.7 - Структура функції ParseZString()

Клас MatrixExt (рис. 3.8) містить 2 функції:

- RowsCount();
- ColumnsCount().

Функція RowsCount() повертає кількість рядків матриці.

Функція ColumnsCount() повертає кількість стовпців матриці.

Функція MatrixPow() повертає матрицю, що є результатом піднесення матриці до степені за допомогою циклу, в якому матриця множиться на ідентичну матрицю p разів, що відповідає числу кількості коефіцієнтів породжувального поліному (рис. 3.9).

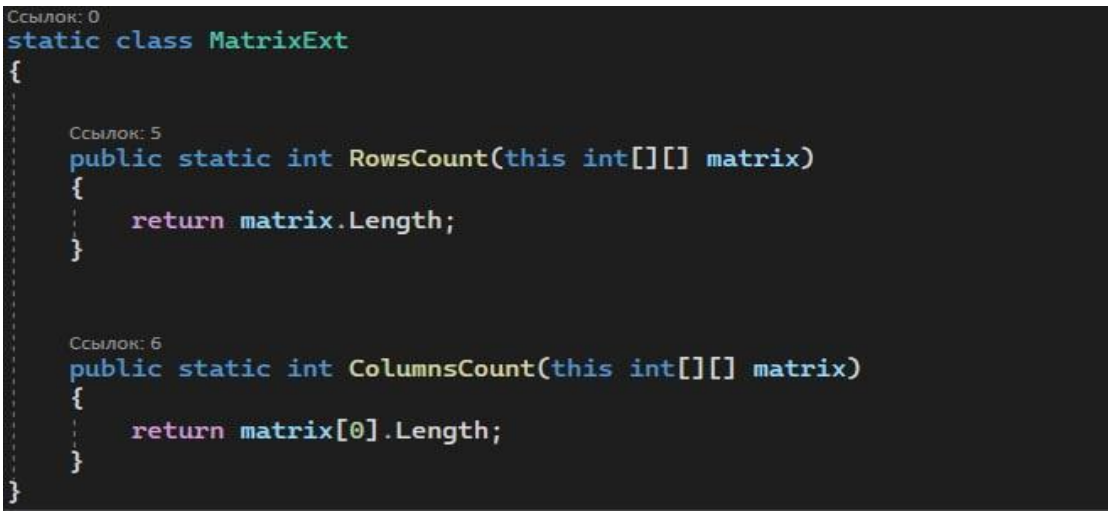


Рисунок 3.8 - Структура класу MatrixExt

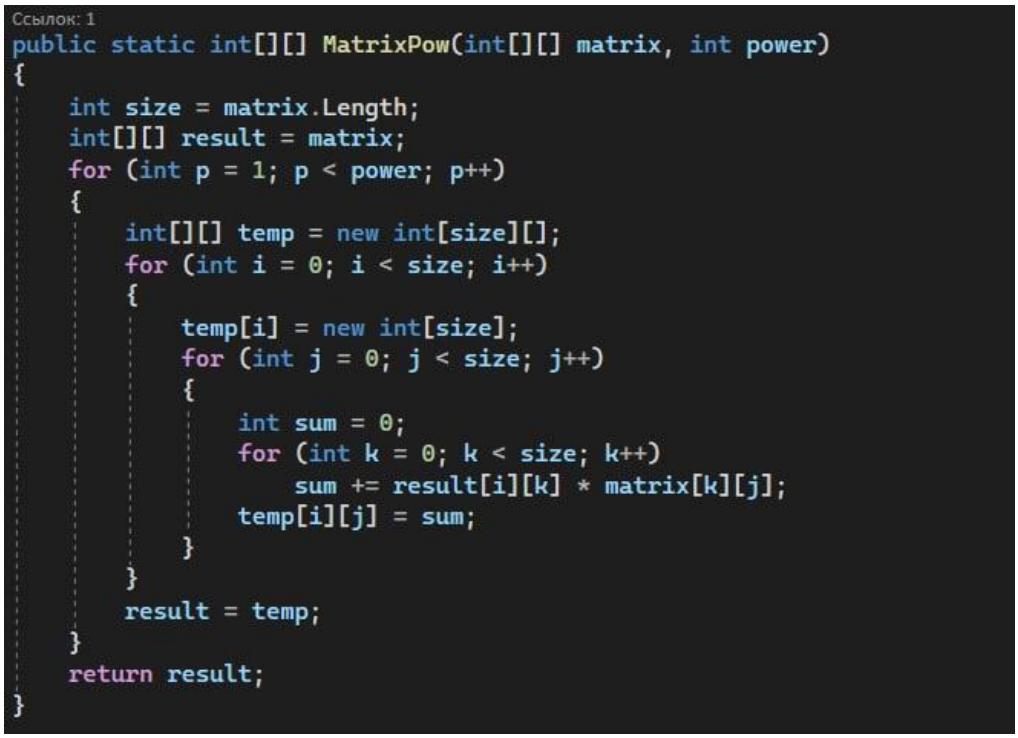


Рисунок 3.9 - Структура функції MatrixPow()

Функція MatrixMultiplication() повертає матрицю, що визначається добутком двох матриць в циклі, але лише у випадку якщо кількість стовпців першої матриці дорівнює кількості рядків другої (рис. 3.10). А інакше, у консолі з’явиться повідомлення «Множення не можливе! Кількість стовпців першої матриці не дорівнює кількості стрічок другої матриці.».

```
Ссылок: 2
public static int[][] MatrixMultiplication(int[][] matrixA, int[][] matrixB)
{
    if (matrixA.ColumnsCount() != matrixB.RowsCount())
    {
        throw new Exception("Умножение не возможно! Количество столбцов первой матрицы не равно количеству строк второй матрицы.");
    }

    var matrixC = new int[matrixA.RowsCount()][];

    for (var i = 0; i < matrixA.RowsCount(); i++)
    {
        matrixC[i] = new int[matrixB.ColumnsCount()];
        for (var j = 0; j < matrixB.ColumnsCount(); j++)
        {
            matrixC[i][j] = 0;

            for (var k = 0; k < matrixA.ColumnsCount(); k++)
            {
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }

    return matrixC;
}
```

Рисунок 3.10 - Структура функції MatrixMultiplication()

Функція MatrixAdd() повертає матрицю, що визначається циклом в якому додаються відповідні комірки двох матриць (рис.3.11).

```
Ссылок: 1
static int[][] MatrixAdd(int[][] matrixA, int[][] matrixB)
{
    var matrixC = new int[matrixA.RowsCount()][];

    for (var i = 0; i < matrixA.RowsCount(); i++)
    {
        matrixC[i] = new int[matrixB.ColumnsCount()];
        for (var j = 0; j < matrixB.ColumnsCount(); j++)
        {
            matrixC[i][j] = matrixA[i][j] + matrixB[i][j];
        }
    }

    return matrixC;
}
```

Рисунок 3.11 - Структура функції *MatrixAdd()*

Функція CalcS() визначає матрицю для кожного стану S по формулі за допомогою циклу, в якому задіяні усі функції, що казана вище(рис.3.12):

$$S_i = A^n \times S_{i-1} + B \times Z_i \qquad (3.1)$$

```
Ссылок: 1
static int[][] CalcS(int[][] a, int n, int[][] sp, int[][] b, int[][] z)
{
    var an = MatrixPow(a, n);
    var ans = MatrixMultiplication(an, sp);
    var bz = MatrixMultiplication(b, z);
    var result = MatrixAdd(ans, bz);

    for (var i = 0; i < result.Length; i++)
    {
        for (var j = 0; j < result[i].Length; j++)
        {
            while (result[i][j] > 1)
            {
                result[i][j] -= 2;
            }
        }
    }

    return result;
}
```

Рисунок 3.12 - Структура функції CalcS()

Алгоритм роботи програми починається з вводу з клавіатури породжувального поліному з коефіцієнтами (рис.3.13).

```
Polinom:
1+1x+0x^2+0x^3+1x^4+0x^5+1x^6+1x^7+x^8
Coeficients: 1, 1, 0, 0, 1, 0, 1, 1
```

Рисунок 3.13 - Введення породжувального поліному.

На основі коефіцієнтів поліному, використовуючи функцію BuildAMatrix(), будується останній стовпчик матриці A, а на основі кількості цих коефіцієнтів вказується розмірність цієї матриці.

За допомогою функції PrintMatrix(), матриця A виводиться на екран у вигляді двомірного масиву (рис. 3.14).

На основі кількості коефіцієнтів поліному, використовуючи функцію BuildBMatrix() будується матриця B з основною діагоналлю, а на основі кількості коефіцієнтів вказується розмірність матриці.

За допомогою функції PrintMatrix(), матриця B виводиться на екран у вигляді двомірного масиву (рис. 3.15).

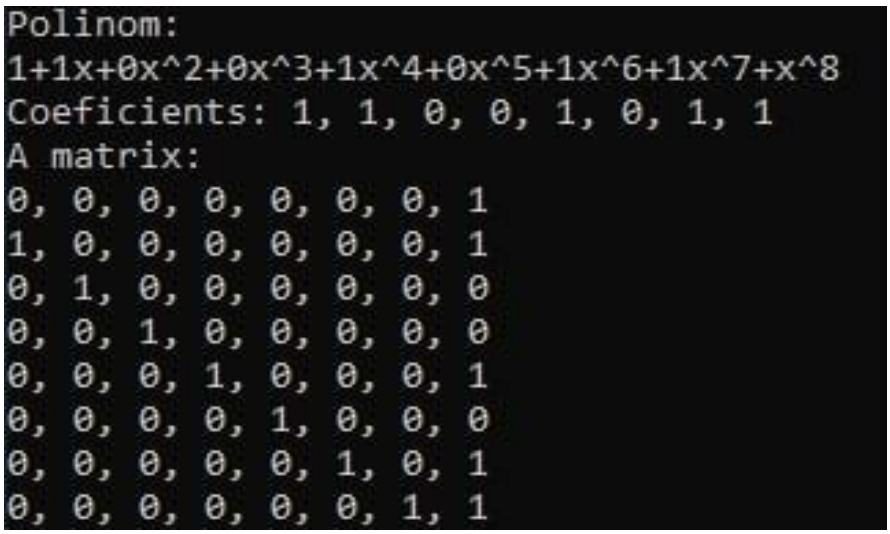


Рисунок 3.14 - Виведення матриці A на екран.

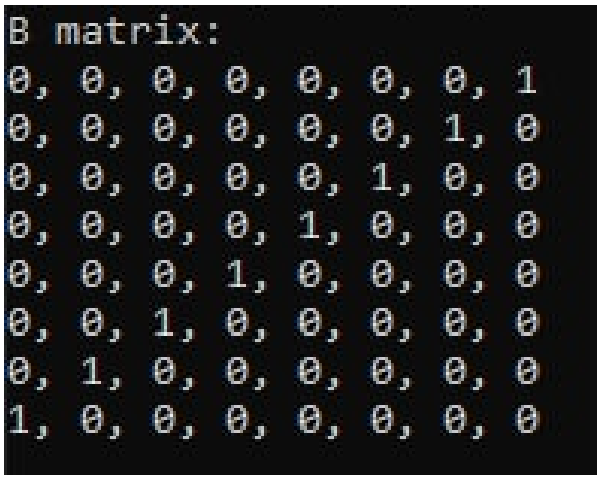


Рисунок 3.15 - Виведення матриці B на екран.

З клавіатури у консоль вводиться вхідна послідовність Z, типу стрічка, без пробілів (рис. 3.16).

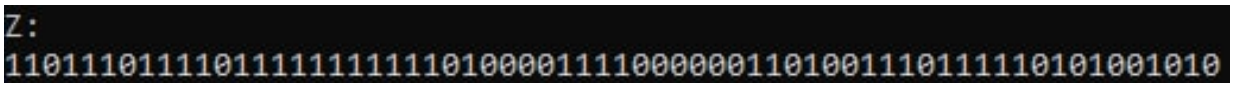


Рисунок 3.16 - Виведення матриці B на екран.

Вміст послідовності обернено записуються у двомірну матрицю Z за

допомогою функції ParseZString().

Матриця Z виводиться в консоль у вигляді двомірного масиву за допомогою функції PrintMatrix() (рис. 3.17).

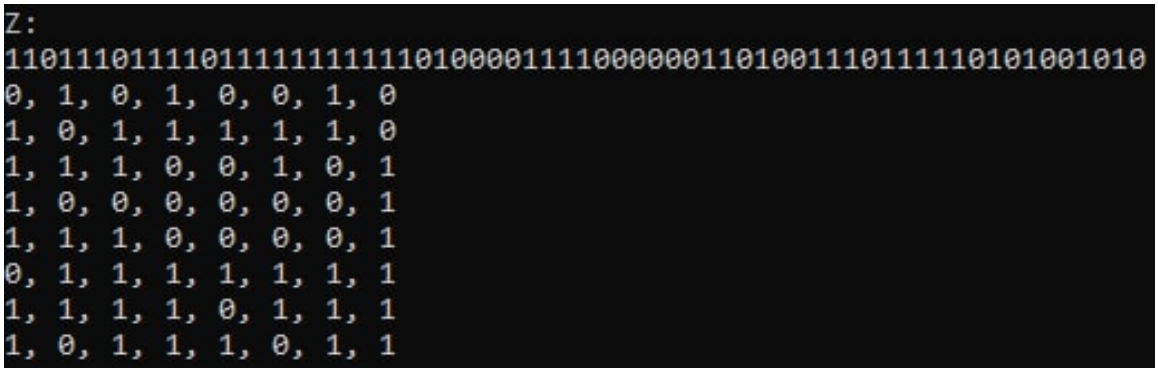


Рисунок 3.17 - Виведення матриці Z на екран.

Проводиться розрахунок матриці першого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю нульового стану за допомогою функції MatrixMultiplication(), тобто на нуль;

2) матриця B множиться на нульову стрічку матриці Z - Z_1 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця першого стану виводиться на екран за допомогою функції PrintMatrix() (рис. 3.17).

Проводиться розрахунок матриці другого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю першого стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на першу стрічку матриці Z - Z_2 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця другого стану виводиться на екран за допомогою функції PrintMatrix() (рис. 3.18).

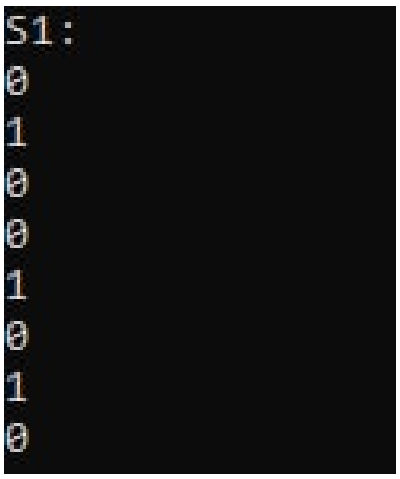


Рисунок 3.17 - Виведення матриці першого стану на екран.

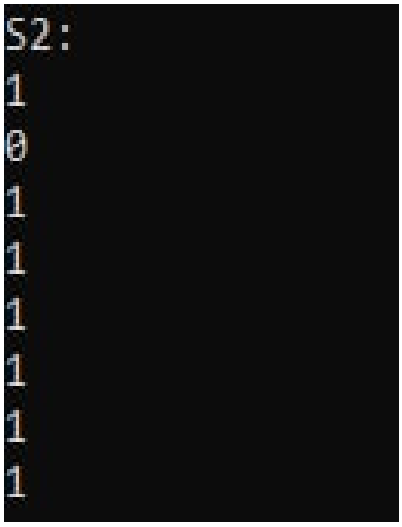


Рисунок 3.18 - Виведення матриці другого стану на екран.

Проводиться розрахунок матриці третього стану, а саме:

1) матриця *A* підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю другого стану за допомогою функції MatrixMultiplication();

2) матриця *B* множиться на другу стрічку матриці Z - Z_3 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця третього стану виводиться на екран за допомогою функції PrintMatrix() (рис 3.19).

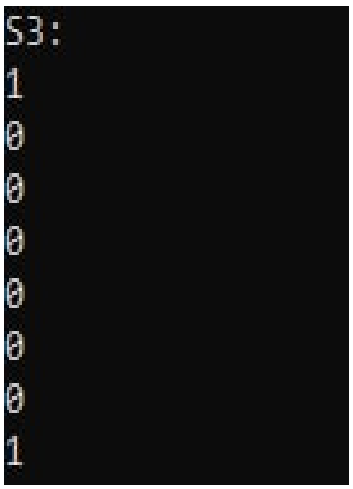


Рисунок 3.19 - Виведення матриці третього стану на екран.

Проводиться розрахунок матриці четвертого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю третього стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на третю стрічку матриці Z - Z_4 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця четвертого стану виводиться на екран за допомогою функції PrintMatrix() (рис 3.20).

Проводиться розрахунок матриці п’ятого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю четвертого стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на четверту стрічку матриці Z - Z_5 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця п’ятого стану виводиться на екран за допомогою функції PrintMatrix() (рис 3.21).

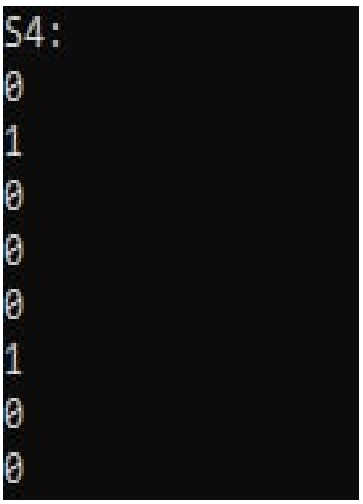


Рисунок 3.20 - Виведення матриці четвертого стану на екран.

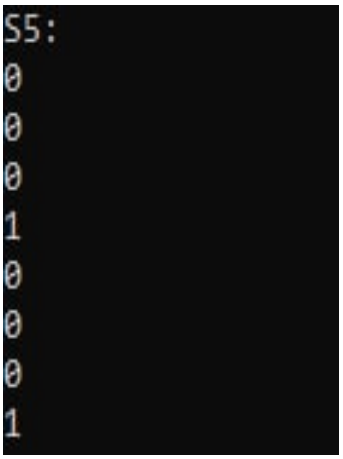


Рисунок 3.21 - Виведення матриці п’ятого стану на екран.

Проводиться розрахунок матриці шостого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю п’ятого стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на п’яту стрічку матриці Z - Z_6 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця шостого стану виводиться на екран за допомогою функції PrintMatrix() (рис 3.22).

Проводиться розрахунок матриці сьомого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю шостого стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на шосту стрічку матриці Z - Z_7 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця сьомого стану виводиться на екран за допомогою функції PrintMatrix() (рис 3.23).

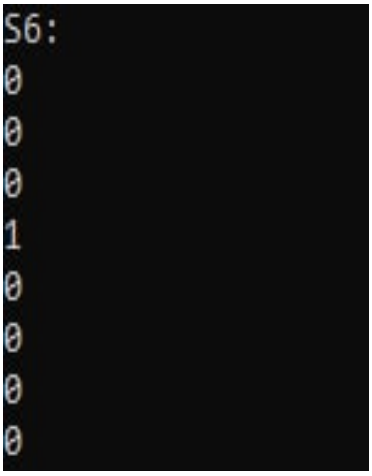


Рисунок 3.22 - Виведення матриці шостого стану на екран.

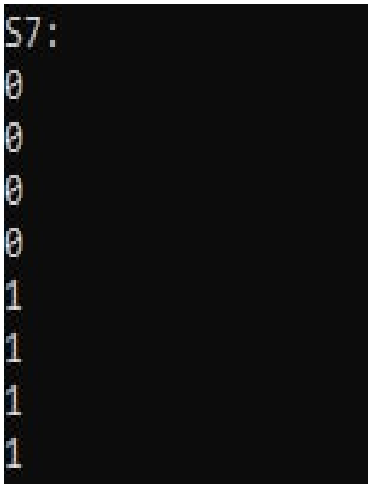


Рисунок 3.23 - Виведення матриці сьомого стану на екран.

Проводиться розрахунок матриці восьмого стану, а саме:

1) матриця A підноситься до степеню кількості коефіцієнтів поліному за допомогою функції MatrixPow(), множиться на матрицю першого стану за допомогою функції MatrixMultiplication();

2) матриця B множиться на сьому стрічку матриці Z - Z_8 за допомогою функції MatrixMultiplication();

3) Результати пунктів 1) і 2) додаються за допомогою функції AddMatrix().

Матриця восьмого стану автомату виводиться на екран в консолі за допомогою функції PrintMatrix(), вона і буде контрольною сумою для даної послідовності (рис. 3.24).

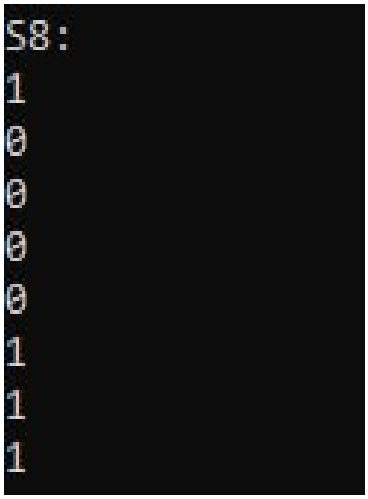


Рисунок 3.19 - Контрольна сума CRC.

Висновки

У першому розділі було розглянуто типи завадостійкого кодування та методи перевірки цілісності, вони є відносно не досконалими, їм не вистачає швидкодії або вони потребують забагато пам’яті. Вони потребують автоматизації і удосконалення взаємодії з користувачем.

У другому розділі розглядалися автоматні методи контролю. Вони забезпечують суттєве прискорення запису інформації на носій та мають більший пріоритет у використанні та удосконаленні, адже проблема запису даних є актуальною проблемою сьогодення. Саме тому, для підвищення швидкості розрахунку контрольної суми запропоновано використовувати математичний апарат багатоканальних ЛПС. Паралелізм цих автоматів дозволяє прискорити обчислення контрольних сум у р разів.

У третьому розділі на основі математичного апарату багатоканальних лінійних автоматів була побудована апаратна схема та програмне рішення за допомогою мови C#.

Перелік джерел посилання

1. Fujiwara E.-// Code Design for Dependable Systems. Theory and Practical Applications. USA: John Willy & Sons, Inc., 2006.

2. Micheloni R., Marelli A., Ravasio R. Error Correction Codes for Non-Volatile Memories. Springer, 2008.

3. Методи виявлення помилок - [Електронний ресурс]. URL: https://lektsii.com/1-35887.html.

5. Дискові масиви RAID - [Електронний ресурс].URL: https://uk.wikipedia.-org/wiki/RAID.

6. Класифікація масивів RAID - [Електронний ресурс] URL: http://ni.biz.-ua/3/3_20/3_-207194_diskovie-massiviRAID.html?msclkid=e0176058b1fa11ecac6-0eb4173e7c925.

7. Надлишкові RAID-масиви - [Електронний ресурс] URL: https://www.uf-sexplorer.com/uk/articles/storage-technologies/raid-data-organization.php.

8. Завадостійке кодування - [Електронний ресурс]. URL : https://helpiks.-org/4197-98.html.

9. Циклічний надлишковий код - [Електронний ресурс]. URL: https://uk.w-ikipedia.org/wiki/Циклічний_надлишковий_код?msclkid=5a1337e6b20111ec890a-b34b01788a5f.

8. Ross N. Williams. - //Элементарное руководство по CRC алгоритмам обнаружения ошибок.

9. Гилл.А - //Лінійні послідовні машини. Аналіз, синтез, використання.

10.Р.Блейхут - //Теорія і практика кодів, що виправляють помилки // Р. Блейхут; пер. с англ. - М.: Мир, 1986. - 576 с.

11. Nguyen, G. D. Fast CRCs / G. D. Nguyen // IEEE Transactions on Computers. - 2009. -Vol. 58, Issue 10. - P. 1321-1331. doi : 10.1109/tc.2009.83.

Додаток А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф.
_____О. Д. Азаров
«__» _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання комплексного бакалаврського дипломного проекту

«Захист в системах зберігання даних на основі контрольних сум CRC»

08-23.БДП.030.00.000 ТЗ

Науковий керівник: доцент к.т.н.

_____ Семеренко В. П.

Виконав: студент групи 2КІ-18б

_____ Степанов О. Д.

Вінниця, 2022

1 Підстава для виконання бакалаврського дипломного проекту (БДП)

1.1 Актуальність розробки полягає у необхідності вирішення проблеми повільного запису цілісних даних на носій, шляхом використання швидкого автоматного методу пошуку контрольних сум CRC.

1.2 Наказ про затвердження теми БДП.

2 Мета БДП і призначення розробки

2.1 Мета роботи - зменшення часу запису та зчитування інформації в комп'ютерних системах зберігання даних.

2.2 Призначення розробки - проаналізовані та отримані дані можна використати для вдосконалення цифрових пристроїв, прискорення запису даних на носії та забезпечення достовірності цих даних.
Запропоновано використання такого захисту даних в дискових масивах RAID.

3 Вихідні дані для виконання БДП

3.1 Проведення аналізу методів пошуку контрольних сум CRC.

3.2 Огляд математичних основ лінійних автоматів.

3.3 Розробка програмного рішення для прискорення пошуку контрольних сум, на основі багатоканальних лінійних автоматів.

4 Вимоги до виконання БДП

4.1 Забезпечити високу швидкодію знаходження контрольної суми.

4.2 Виконати програмне рішення, використовуючи мову програмування C#.

4.3 Забезпечити підтримку вісьми і більше розрядів поліному.

5 Етапи БДП та очікувані результати

Етапи проекту та очікувані результати приведено в Таблиці А.1.

6 Матеріали, що подаються до захисту БДП

До захисту подаються: пояснювальна записка БДП, програмне рішення, протокол попереднього захисту БДП на кафедрі, відзив наукового керівника,

рецензія опонента, анотації до БДП українською та іноземною мовами, довідка про відповідність оформлення БДП діючим вимогам.

Таблиця А.1 - Етапи БДП

№	Назва	Термін		Результат
		початок	кінець	
1	Постановка задачі роботи	07.03.22		Вступ Розділ 1
2	Пошук матеріалів про кодування інформації	08.03.22	31.03.22	Розділ 1
3	Пошук та аналіз типів завадостійкого кодування	04.04.22	15.04.22	Розділ 1
4	Аналіз математичних основ контрольних сум	18.04.22	22.04.22	Розділ 1
5	Аналіз розповсюджених методів знаходження контрольних сум CRC	25.04.22	06.05.22	Розділ 1
6	Аналіз математичного апарату лінійних автоматів	09.05.22	13.05.22	Розділ 2
7	Аналіз автоматних методів пошуку контрольної суми	16.05.22	20.05.22	Розділ 2
8	Підготовка програмного та апаратного рішення	23.05.22	26.05.22	Розділ 3
9	Аналіз виконання роботи, висновки, додатки	27.05.22	31.05.22	Пояснювальна записка
10	Перевірка якості виконання бакалаврського проекту та усунення недоліків	01.06.22	08.06.22	Пояснювальна записка, презентація

7 Порядок контролю виконання та захисту БДП

Виконання етапів програмної та розрахункової документації БДП контролюється науковим керівником згідно зі встановленими термінами. Захист БДП відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення БДП

При оформлюванні БДП використовуються:

- ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

- ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

- ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

- документами на які посилаються у вище вказаних.

Технічне завдання до виконання отримав _____ Степанов О. Д.

Додаток Б

Ліснінг програми

```
using System;
using System.Linq;
using System.Text.RegularExpressions;

namespace Adder
{
internal class Program
{
public static void Main()
{
Console.WriteLine("Polinom: ");
var polinomSting = Console.ReadLine();
var polinomCoeficients = Regex.Matches(polinomSting, @"(?<!\^)[0-9]+")
.Select(x => int.Parse(x.Value))
.SkipLast(1)
.ToArray();
Console.WriteLine($"Coeficients: {string.Join(", ", polinomCoeficients)}");
Console.ReadKey();
var size = polinomCoeficients.Length;
Console.WriteLine("A matrix:");
var aMatrix = BuildAMatrix(polinomCoeficients);
PrintMatrix(aMatrix, size);
Console.ReadKey();
Console.WriteLine("B matrix:");
var bMatrix = BuildBMatrix(size);
PrintMatrix(bMatrix, size);
Console.ReadKey();
```

```
Console.WriteLine("Z: ");
var zString = Console.ReadLine();
var z = ParseZString(zString, size);
PrintMatrix(z, size);
Console.ReadKey();
var sMatrix = new int[size][];
for (int i = 0; i < size; i++)
{
sMatrix[i] = new int[1];
sMatrix[i][0] = 0;
}
for (int i = 0; i < size; i++)
{
var zMatrix = new int[size][];
for (int j = 0; j < size; j++)
{
zMatrix[j] = new int[1];
zMatrix[j][0] = z[i][j];
}
sMatrix = CalcS(aMatrix, size, sMatrix, bMatrix, zMatrix);
Console.WriteLine($"S{i + 1}: ");
PrintMatrix(sMatrix, size);
Console.ReadKey();
}
}
public static int[][] BuildAMatrix(int[] polinom)
{
var result = new int[polinom.Length][];
for (int i = 0; i < polinom.Length; i++)
```

```
{
result[i] = new int[polinom.Length];
for (int j = 0; j < polinom.Length; j++)
{
if (i == j + 1)
{
result[i][j] = 1;
}
else if (j == polinom.Length - 1)
{
result[i][j] = polinom[i];
}
else
{
result[i][j] = 0;
}
}
}

return result;
}

public static int[][] BuildBMatrix(int size)
{
var result = new int[size][];
for (int i = 0; i < size; i++)
{
result[i] = new int[size];
for (int j = 0; j < size; j++)
```

```
{
if (i == size - j - 1)
{
result[i][j] = 1;
}
else
{
result[i][j] = 0;
}
}
}
return result;
}
public static void PrintMatrix(int[][] matrix, int size)
{
for (int i = 0; i < size; i++)
{
Console.WriteLine(string.Join(", ", matrix[i]));
}
}
public static int[][] ParseZString(string zString, int size)
{
var result = new int[size][];
for (int i = 0; i<size; i++)
{
result[i] = new int[size];
for (int j = 0; j <size; j++)
{
result[i][j] = int.Parse(zString[(size - 1 - i) * size + (size - 1 - j)].ToString());
```

```
}
}
return result;
}
public static int[][] MatrixPow(int[][] matrix, int power)
{
int size = matrix.Length;
int[][] result = matrix;
for (int p = 1; p < power; p++)
{
int[][] temp = new int[size][];
for (int i = 0; i < size; i++)
{
temp[i] = new int[size];
for (int j = 0; j < size; j++)
{
int sum = 0;
for (int k = 0; k < size; k++)
sum += result[i][k] * matrix[k][j];
temp[i][j] = sum;
}
}
result = temp;
}
return result;
}
public static int[][] MatrixMultiplication(int[][] matrixA, int[][] matrixB)
{
if (matrixA.ColumnsCount() != matrixB.RowsCount())
```

```
{
throw new Exception("Умножение не возможно! Количество столбцов первой матрицы не равно количеству строк второй матрицы.");
}
var matrixC = new int[matrixA.RowsCount()][];
for (var i = 0; i < matrixA.RowsCount(); i++)
{
matrixC[i] = new int[matrixB.ColumnsCount()];
for (var j = 0; j < matrixB.ColumnsCount(); j++)
{
matrixC[i][j] = 0;
for (var k = 0; k < matrixA.ColumnsCount(); k++)
{
matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
}
}
}
return matrixC;
}
static int[][] MatrixAdd(int[][] matrixA, int[][] matrixB)
{
var matrixC = new int[matrixA.RowsCount()][];
for (var i = 0; i < matrixA.RowsCount(); i++)
{
matrixC[i] = new int[matrixB.ColumnsCount()];
for (var j = 0; j < matrixB.ColumnsCount(); j++)
{
matrixC[i][j] = matrixA[i][j] + matrixB[i][j];
}
```

```
}
return matrixC;
}
static int[][] CalcS(int[][] a, int n, int[][] sp, int[][] b, int[][] z)
{
var an = MatrixPow(a, n);
var ans = MatrixMultiplication(an, sp);
var bz = MatrixMultiplication(b, z);
var result = MatrixAdd(ans, bz);
for (var i = 0; i < result.Length; i++)
{
for (var j = 0; j < result[i].Length; j++)
{
while (result[i][j] > 1)
{
result[i][j] -= 2;
}
}
}
return result;
}
}
static class MatrixExt
{public static int RowsCount(this int[][] matrix)
{
return matrix.Length;
}
public static int ColumnsCount(this int[][] matrix)
{ return matrix[0].Length;}}}
```

Додаток В

Блок-схема алгоритму функції ParseZString()

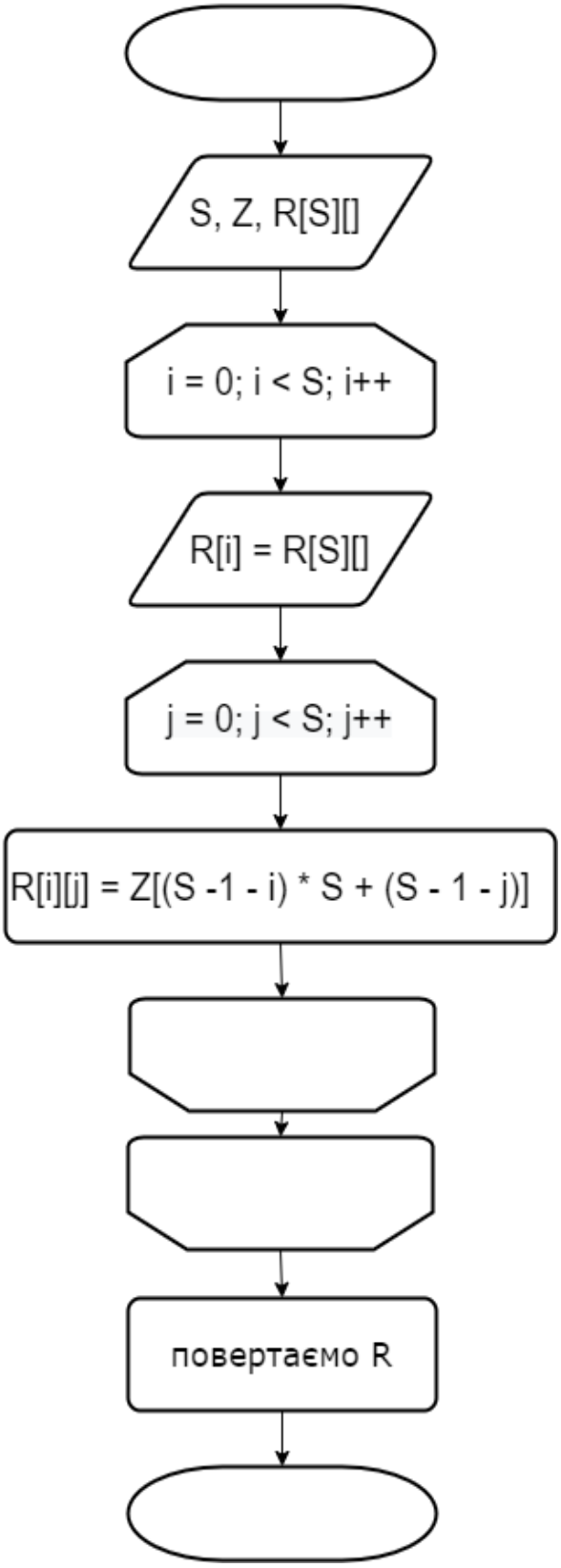


Рисунок В.1 - Блок-схема алгоритму ParseZString()

Додаток Г

Блок-схема алгоритму функції PrintMatrix()

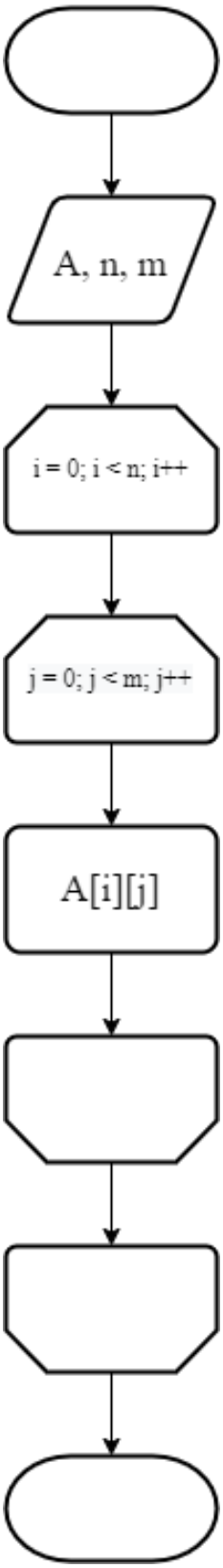


Рисунок Г.1 - Блок-схема алгоритму PrintMatrix()

Додаток Д

Блок-схема алгоритму функції MatrixPow()

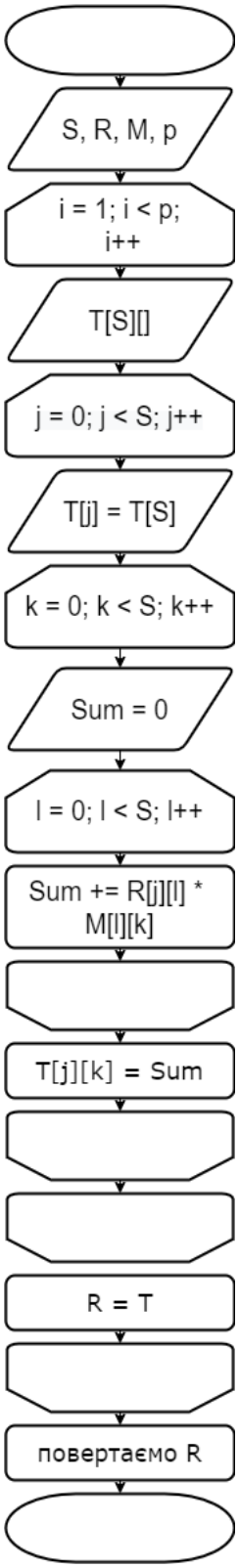


Рисунок Д.1 - Блок-схема алгоритму MatrixPow()

Додаток Е

Блок-схема алгоритму функції BuiltAMatrix()

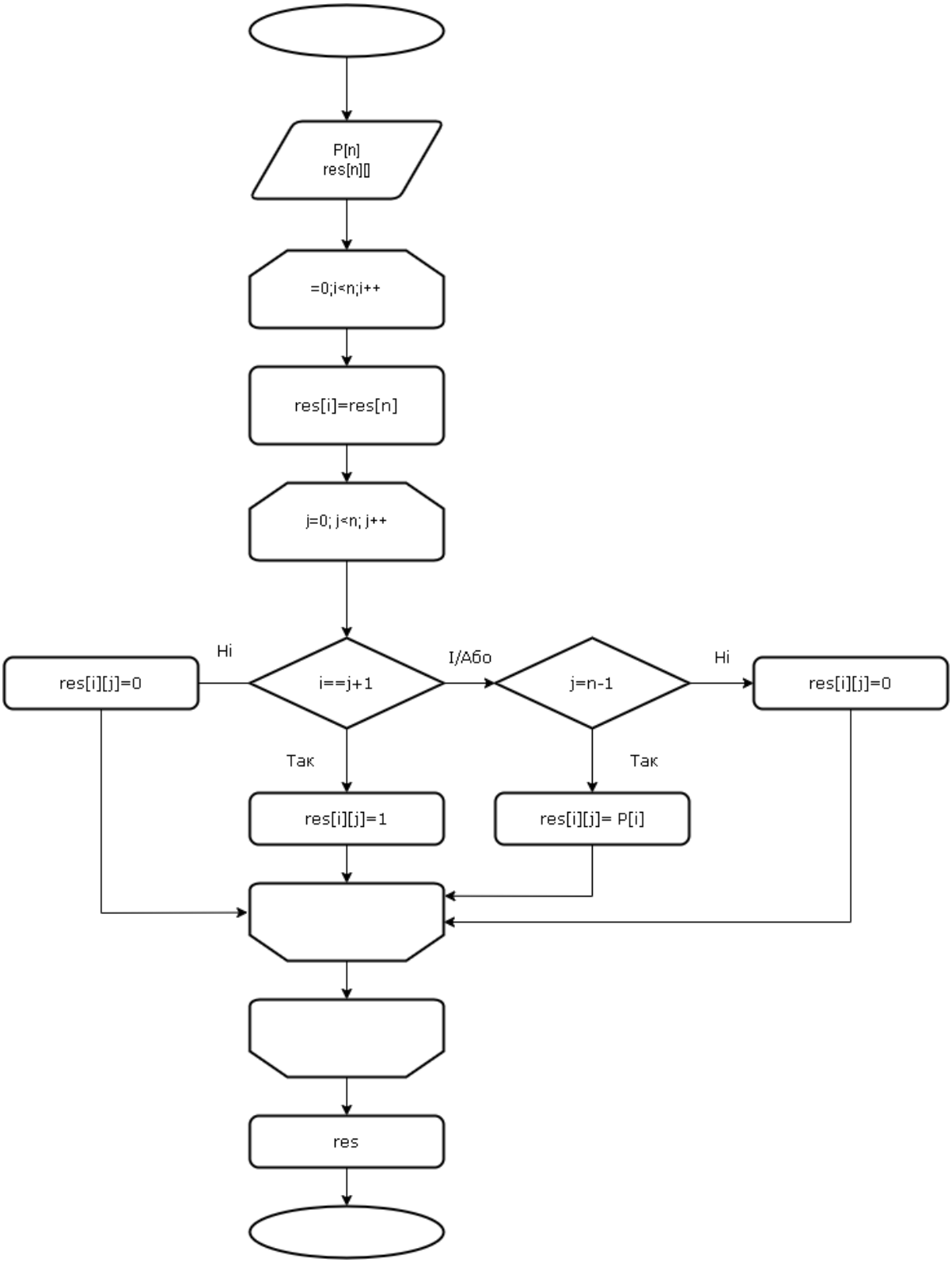


Рисунок Е.1 - Блок-схема алгоритму BuiltAMatrix()

Додаток Ж

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Захист в системах зберігання даних на основі контрольних сум в CRC

Тип роботи: бакалаврський дипломний проект

Підрозділ кафедра обчислювальної техніки

Показники звіту подібності
Unicheck

Оригінальність 74,3% Схожість 25,7%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Степанов О. Д.

Керівник роботи _____ Семеренко В. П.