

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра інформатики та інтелектуальної власності  
Спеціальність комп'ютерні науки  
Освітня програма комп'ютерні науки

**КУРСОВА РОБОТА**  
**пояснювальна записка**

на тему

«Розробка алгоритмів та прикладних програм моделювання генераторів  
псевдовипадкових послідовностей»

з дисципліни «Аналіз та проектування алгоритмів»

Виконав студент 3 курсу  
групи КН-319в

Оніщенко Д. П.

Керівник

проф. Солощук М.М.

Оцінка

\_\_\_\_\_ (національна)

\_\_\_\_\_ (бал)

\_\_\_\_\_ (ECTS)

Харків 2022

Національний технічний університет  
«Харківський політехнічний інститут»

Кафедра інформатики та інтелектуальної власності

Дисципліна: Аналіз та проектування алгоритмів

Студент :        Оніщенко Д. П.                    Група    КН – 319В                    Курс: 3

ЗАВДАННЯ

на курсовий проект з навчальної дисципліни

"Аналіз та проектування алгоритмів"

Постановка задачі: При виконанні курсового проекту необхідно спроектувати алгоритми та реалізувати програми моделювання генераторів псевдовипадкових послідовностей:

1. Генератора, побудованого на базі зсувного регістру з лінійними зворотними зв'язками (ЗРЗЗ) , що описується векторна-матричним рекурентним рівнянням над полем  $GF(2)$  виду:

$$Y_{[I+1]} = F \cdot Y_{[I]}$$

де

$I = \{0, 1, 2, 3, \dots\}$ ;

$F$  – квадратна структурна матриця ЗРЗЗ розміру  $n \cdot n$ ;

$Y_{[0]}$ ;  $Y_{[I]}$  та  $Y_{[I+1]}$  – вектори початкового, попереднього та наступного станів ЗРЗЗ розміру  $n$ .

2. Генератора, побудованого на базі матричного зсувного регістру (МЗР), який є реалізацією автономної матричної лінійної послідовнісної машини (АМЛПМ), що описується матричним рекурентним рівнянням над полем  $GF(2)$  виду:

$$S_{[I+1]} = A \cdot S_{[I]} \cdot B$$

де

$$I = \{0, 1, 2, 3, \dots\};$$

$A$  і  $B$  – квадратні структурні матриці АМЛПМ розміру  $n \cdot n$  та  $m \cdot m$  відповідно;  
 $S_{[0]}$ ;  $S_{[I]}$  та  $S_{[I+1]}$  – матриці початкового, попереднього та наступного станів МЗР (АМЛПМ) розміру  $n \cdot m$ .

Програми моделювання повинні складатися з клієнтської та серверної частин. Серверна частина (backend) повинна забезпечувати взаємодію з сервером бази даних нерозкладних поліномів, дозволяти обчислювати послідовність станів ЗРЗЗ і МЗР, визначати періоди матриць  $F$ ,  $A$ ,  $B$  і/або періоди векторів-станів  $Y$  і матриць-станів  $S$ , формувати двійкові послідовності станів заданого елементу вектору-стану  $Y_{[I]}$  і/або матриці-стану  $S_{[I]}$  при  $I = \{0, 1, 2, 3, \dots\}$ , обчислювати вагові функції Хеммінгу цих послідовностей та періодичні автокореляційні функції послідовностей з елементами  $\pm 1$ , побудованих з цих двійкових послідовностей (технології: C++, Java, Python, JavaScript та інші).

Повинно бути реалізоване введення та відображення вихідних даних (ступенів поліномів  $n$  та  $m$ , характеристичних поліномів матриць  $F$ ,  $A$  і  $B$ , початкових станів  $Y_{[0]}$  і  $S_{[0]}$ ), а також візуалізація результатів моделювання за допомогою інтерфейсу користувача (frontend - технології html, CSS, JavaScript). Повинне бути забезпечене редагування та збереження бази даних варіантів вихідних даних, а також генерація та збереження звітів про результати моделювання генератора для вибраних варіантів вихідних даних. Програма має бути розділена на модулі.

### 3. Варіативна частина завдання:

3.1. Тема реферату-огляду обирається відповідно до Додатку 1.

3.2. Додаткове завдання з моделювання обирається відповідно до Додатку 2.

### 4. Короткий зміст роботи:

а) Реферативна частина

- Реферат-огляд періодичної науково-технічної літератури та вивчення методів та алгоритмів заданого класу у вигляді текстового документу, схем, малюнків та презентації реферату-огляду (частина І)

б) Теоретична частина

- Формалізація завдання, опис використовуваних математичних методів
- Розробка архітектури та функціональності програмної системи
- Проектування, аналіз ефективності та оптимізація алгоритмів моделювання генераторів

в) Практична частина

- Розробка функціональної схеми модельованого пристрою
- Обґрунтування вибору програмних засобів, розробка програмної реалізації алгоритмів, що моделюють пристрій
- Експериментальне дослідження алгоритмів
- Оформлення результатів досліджень та формулювання висновків
- Оформлення пояснювальної записки, схем, програмної документації та презентації

г) захист курсового проекту у вигляді презентації (частина ІІ) результатів за пунктами б) та в)

Дата видачі  
завдання:

04 травня 2022

Термін захисту: 21 – 28 травня 2022

Керівник курсової роботи:

Студент:

проф. Солощук М.М.



Оніщенко Д. П.

(Підпис)

(ПІБ)

(Підпис)

(ПІБ)

## РЕФЕРАТ

Пояснювальна записка: 37с., 21рис., 9джерел.

Ключові слова: тестування псевдовипадкових послідовностей, сигнатурний аналіз, signature scheme with appendix-probabilistic signature scheme, лінійний зсувний регістр, матричний зсувний регістр.

Об'єкт дослідження – генератори псевдовипадкових чисел.

Мета роботи – дослідження та розробка алгоритмів генерації псевдовипадкових чисел.

Методи та технології дослідження й розробки – метод імперативного та об'єктно орієнтованого програмування та методи математичної індукції.

Результат роботи – збір теоретичних відомостей про формування використання псевдовипадкових чисел; математична постановка задачі генерації псевдовипадкових чисел за допомогою зсувних регістрів; програмна розробка продукту; тестування продукту.

## ЗМІСТ

ВСТУП.....	6
1 Псевдовипадкове тестування кодів, цифрових схем та комп'ютерних програм	7
1.1 Поняття псевдовипадкових послідовностей.....	7
1.2 Тестування псевдовипадкових послідовностей. ....	11
1.3 Сигнатурний аналіз. ....	12
1.4 Цифровий підпис .....	14
1.5 RSASSA-PSS .....	15
2 Формалізація поставленої задачі.....	19
2.1 Математична постановка задачі.....	19
2.2 Розробка архітектури та функціональності програмної системи.....	23
3 Програмування застосунку .....	25
3.1 Обрані технології розробки .....	25
3.2 Розробка LFSR .....	25
3.3 Розробка MSR .....	26
3.4 Розробка інтерфейсу користувача.....	28
4 Тестування розробленого продукту.....	29
4.1 Тестування LFSR .....	29
4.2 Тестування MSR .....	32
ВИСНОВКИ.....	35
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	36

## ВСТУП

Теорія псевдовипадкових величин та псевдовипадкових послідовностей має безліч різних термінів, задач та методів. Попри простоту формування задачі генерування випадкових чисел є дуже складною задачею як з точки зору математики та і апаратних або програмних рішень. На сьогоднішній день кожна мова програмування містить якийсь модуль, котрий відповідає за генерацію випадкових чисел, проте чи дійсно вони є випадковими? Звісно що ні. Основною проблемою генерації випадкових чисел є відсутність випадковості як токової у природі.

Метою роботи є дослідження сфер використання псевдовипадкових чисел та створення програмної реалізації для лінійних та матричних зсувних регістрів. У роботі описуються теоретичні відомості про випадкові та псевдовипадкові числа, послідовності та генератори псевдовипадкових послідовностей.

Основним результатом роботи буде створення програмного додатку для моделювання лінійних зсувних регістрів зі зворотними зв'язками та матричного зсувного регістру. Також додатковий функціонал для визначення певних статистик псевдовипадкових бінарних послідовностей, котрі отримано з розроблених генераторів.

# 1 ПСЕВДОВИПАДКОВЕ ТЕСТУВАННЯ КОДІВ, ЦИФРОВИХ СХЕМ ТА КОМП'ЮТЕРНИХ ПРОГРАМ

## 1.1 Поняття псевдовипадкових послідовностей.

Для більш детального поглиблення у тему слід зазначити декілька термінів та зрозуміти їх суть. Слід почати з термінів псевдовипадкового числа, псевдовипадкової послідовності, бінарної псевдовипадкової послідовності та генератору псевдовипадкових послідовностей.

Джерело справжніх випадкових чисел знайти доволі складно. Щоб генерація чисел була дійсно випадковою необхідно використовувати фізичні явища, наприклад – шуми. Прикладами можуть бути: детектори подій іонізуючої радіації, дробовий шум у резисторі чи космічне випромінювання[1]. Однак застосовуються такі пристрої у програмах мережевої безпеки доволі не часто. Складнощі також викликають грубі атаки на подібні пристрої.

У фізичних джерел випадкових чисел існує низка недоліків:

- Час та трудовитрати при встановленні та налаштуванні порівняно з програмними засобами;
- Ціна;
- Генерація випадкових чисел відбувається повільніше, ніж за програмної реалізації;
- Неможливість відтворення раніше генеровані послідовності випадкових чисел.

У той самий час випадкові числа, одержувані з фізичного джерела, можна використовувати як формуючий елемент (seed) для програмних генераторів. Такі комбіновані генератори використовуються в криптографії, лотереях, ігрових автоматах[2].

Псевдовипадкова послідовність – послідовність чисел, яка була обчислена за деяким певним арифметичним правилом, але має всі властивості випадкової послідовності чисел у рамках вирішуваного завдання.



Хоча псевдовипадкова послідовність у цьому сенсі часто, як здається, позбавлена закономірностей, проте будь-який псевдовипадковий генератор з кінцевим числом внутрішніх станів повториться після дуже довгої послідовності чисел. Це можна довести з допомогою принципу Дирихле.

Псевдовипадкова двійкова послідовність – окремий випадок псевдовипадкової послідовності, в якій елементи приймають два можливі значення 0 і 1 (або -1 і +1). Такі послідовності є періодичними.

Одне з перших формулювань деяких основних правил для статистичних властивостей періодичних псевдовипадкових послідовностей було представлено Соломоном Голомбом. Три основні правила здобули популярність як постулати Голомба:

1. Кількість "1" у кожному періоді повинна відрізнятись від кількості "0" не більше, ніж на одиницю.
2. У кожному періоді половина серія повинна мати довжину один, одна чверть повинна мати довжину два, одна восьма повинна мати довжину три тощо. Більш того, для кожної з цих довжин має бути однакова кількість серій "1" і "0".
3. Припустимо, ми маємо дві копії однієї і тієї ж послідовності періоду  $p$ , зрушені відносно один одного на деяке значення  $d$ . Тоді для кожного  $d$ ,  $0 \leq d \leq p - 1$ , ми можемо підрахувати кількість узгодженостей між цими двома послідовностями  $A_d$  і кількість неузгодженостей  $D_d$ . Коефіцієнт автокореляції для кожного  $d$  визначається співвідношенням  $\frac{A_d - D_d}{p}$  і ця функція автокореляції набуває різних значень у міру того, як  $d$  проходить всі допустимі значення. Тоді для будь-якої послідовності, яка відповідає правилу 3, автокореляційна функція повинна приймати лише два значення.

Постулат 3 – це технічний вираз того, що Голомб описав як поняття незалежних випробувань: знання певного попереднього значення послідовності, в принципі, не допомагає припущенням про поточне значення. Ще одна думка на АКФ у тому, що це певна міра можливості, що дозволяє розрізняти послідовність та її копію, але що починається у деякій іншій точці циклу.

Послідовність, що задовольняє постулатам Голомба, часто називається псевдо-шумовою послідовністю або ПШ-послідовністю. До аналізованої послідовності застосовується широкий спектр різних статистичних тестів на дослідження того, наскільки добре вона узгоджується з припущенням, що з генерації використовувався цілком випадкове джерело.

Генератор псевдовипадкових чисел – алгоритм, що породжує послідовність чисел, елементи якої майже незалежні один від одного і підпорядковуються заданому розподілу. Сучасна інформатика широко використовує псевдовипадкові числа в різних додатках – від методу Монте-Карло та імітаційного моделювання до криптографії. При цьому від якості використовуваних генераторів безпосередньо залежить якість результатів.

Зазвичай генератори псевдовипадкових чисел повинні мати наступні характеристики:

1. Достатня довжина періоду. Щоб числа не почались повторюватись дуже рано;
2. Ефективність. Генерація наступного випадкового числа повинна виконуватись дуже швидко;
3. Повторність. Тобто в залежності від початкових показників повинна бути можливість відтворити таку саму послідовність;
4. Можливість порту алгоритму. Реалізація та робота алгоритму не повинна залежати від архітектури операційної системи або апаратної засобів.

Регістр зсуву з лінійним зворотним зв'язком (англ. linear feedback shift register, LFSR, Рис. 1.1) – зсувний регістр бітових слів, у якого значення вхідного біта дорівнює лінійної булевої функції від значень інших бітів регістру до зсуву. Можливо організований як програмними, і апаратними засобами. Застосовується для генерації псевдовипадкових послідовностей бітів. За схожим принципом працюють регістр зсуву зі зворотним зв'язком по перенесенню та регістр зсуву із узагальненим зворотним зв'язком. Завдяки своїй будові, вони легко піддаються аналізу за допомогою алгебраїчних технік.

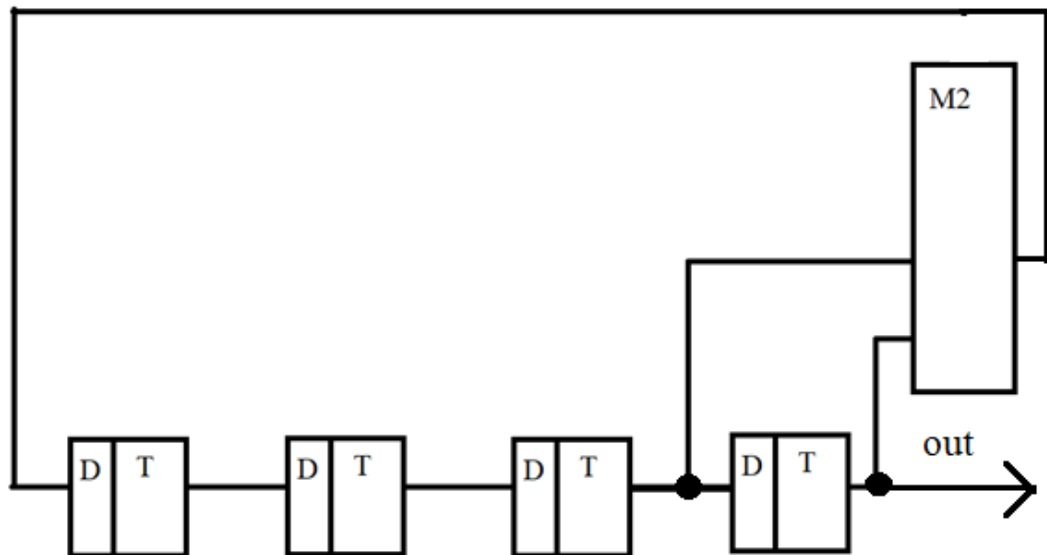


Рисунок 1.1 – Базова схема лінійного зсувного реєстру зі зворотними зв'язками

Як вже було сказано, лінійні та інші види генераторів псевдовипадкових чисел використовуються для формування псевдовипадкових числових послідовностей. Одним з прикладом таких послідовностей є М-послідовності.

М-послідовність або послідовність максимальної довжини – псевдовипадкова двійкова послідовність, породжена регістром зсуву з лінійним зворотним зв'язком і має максимальний період. М-послідовності застосовують у широкосмугових системах зв'язку. Така послідовність має декілька дуже важливих характеристик:

- М-послідовності є періодичними з періодом  $T = 2^n - 1$ ;
- кількість символів, що набирають значення одиниця, на довжині одного періоду М-послідовності на одиницю більше, ніж кількість символів, що набувають значення нуль;
- будь-які комбінації символів довжини  $n$  на довжині одного періоду М-послідовності за винятком комбінації з  $n$  нулів зустрічаються не більше одного разу. Комбінація з  $n$  нулів є забороненою: її основі може генеруватися лише послідовність з одних нулів;
- сума за модулем 2 будь-якої М-послідовності з її довільним циклічним зрушенням також М-послідовністю;

- періодична АКФ будь-якої  $M$ -послідовності має постійний рівень бічних пелюсток, що дорівнює  $\frac{1}{N}$ [3];
- АКФ усіченої  $M$ -послідовності, під якою розуміється неперіодична послідовність довжиною в період  $N$ , має величину бічних пелюсток, близьку до  $\frac{1}{\sqrt{N}}$ . Тому зі зростанням  $N$  величина бічних піків зменшується.

## 1.2 Тестування псевдовипадкових послідовностей.

Тестування псевдовипадкових послідовностей – сукупність методів визначення міри близькості заданої псевдовипадкової послідовності до випадкової. Як такий захід зазвичай виступає наявність рівномірного розподілу, великого періоду, рівної частоти появи однакових підрядків тощо.

Один із тестів, при котрому одразу видно результат – тест на рівномірний розподіл частот появи кожного символу. Нехай  $\xi_1, \xi_2, \xi_3$  – послідовність довжиною  $n$  та розмірності  $m$ . Тоді частоти повинні належати відрізка:

$$\left[ \frac{n - 2.58 \sqrt{n(2^m - 1)}}{2^m}; \frac{n + 2.58 \sqrt{n(2^m - 1)}}{2^m} \right]$$

Проте, більшість тестів використовують інший метод – прийняття чи відхилення гіпотези про випадковість послідовності, використовуючи статистичні розподіли. Знаючи ймовірні властивості істинно випадкової послідовності, можна на їх основі перевіряти гіпотезу про те, наскільки випадкова послідовність схожа на дійсно випадкову. Для цього для кожного тесту підбирається підходяща статистика, обчислюється її значення для ідеальної та даної послідовності. Якщо різниця цих значень перевищує деяке критичне значення, встановлене заздалегідь, послідовність вважається не випадковою. Визначимо величину  $P$  – *value* як ймовірність того, що ідеальний генератор згенерував менш випадкову послідовність, ніж досліджуваний. Тоді якщо  $P$  – *value* більше  $\alpha$ , то досліджувана послідовність вважається випадковою і відповідно навпаки.

### 1.3 Сигнатурний аналіз.

Сигнатурний антивірусний аналіз – це один із методів антивірусного захисту, що полягає у виявленні характерних ідентифікуючих властивостей кожного вірусу та пошуку вірусів при порівнянні файлів із виявленими властивостями. Однією з найважливіших властивостей сигнатурного аналізу є точне визначення типу вірусу. Це дозволяє занести основу як сигнатури так і способи лікування вірусу.

Виявлення, засноване на сигнатурах – метод роботи антивірусів і систем виявлення вторгнень, у якому програма, переглядаючи файл чи пакет, звертається до словника з відомими вірусами, складеному авторами програми. У разі відповідності будь-якої ділянки коду програми, що переглядається відомому коду (сигнатурі) вірусу в словнику, програма антивірус може зайнятися виконанням однієї з наступних дій: видалити інфікований файл; надіслати файл до «карантину», тобто зробити його недоступним для виконання з метою недопущення подальшого поширення вірусу, спробувати відновити файл, видаляючи сам вірус з тіла файлу.

Для досягнення тривалого успіху під час використання цього методу необхідно періодично поповнювати словник відомих вірусів новими визначеннями. Антивірусні програми, створені на основі методу відповідності визначення вірусів у словнику, зазвичай переглядають файли тоді, коли комп'ютерна система створює, відкриває, закриває або надсилає файли електронною поштою. Таким чином, віруси можна виявити відразу після занесення їх у комп'ютер і до того, як вони зможуть заподіяти будь-яку шкоду. Треба відзначити, що системний адміністратор може скласти графік антивірусної програми, згідно з яким можуть проглядатися (скануватися) всі файли на жорсткому диску.

Хоча антивірусні програми, створені на основі пошуку відповідності визначенню вірусу в словнику, за звичайних обставин можуть досить ефективно перешкоджати спалахам зараження комп'ютерів, автори вірусів намагаються триматися на півкроку попереду таких програм-антивірусів, створюючи віруси, в яких деякі частини коду перезаписуються, модифікуються, шифруються або

спотворюються так, щоб неможливо було виявити збіг з визначенням у словнику вірусів.

Сигнатури антивірусів створюються в результаті копіткого аналізу кількох копій файлу, що належить одному вірусу. Сигнатура повинна містити лише унікальні рядки з цього файлу, настільки характерні, щоб гарантувати мінімальну можливість помилкового спрацювання – головний пріоритет будь-якої антивірусної компанії. Розробка сигнатур – ручний процес, що важко піддається автоматизації. Незважаючи на масу досліджень, присвячених автоматичній генерації сигнатур, наростаючий поліморфізм вірусів та атак роблять синтаксичні сигнатури безглуздими. Антивірусні компанії змушені випускати велику кількість сигнатур для всіх варіантів того самого вірусу, і якби не закон Мура, жоден сучасний комп'ютер вже не зміг би закінчити сканування великої кількості файлів з такою масою сигнатур у розумний час. Так, у березні 2006 року сканеру Norton Antivirus було відомо близько 72 131 вірусів, а база програми містила близько 400 000 сигнатур. У нинішньому вигляді бази сигнатур повинні поповнюватися регулярно, оскільки більшість антивірусів не в змозі виявляти нові віруси самостійно. Будь-який власник ПЗ, заснованого на сигнатурах, приречений на регулярну залежність від оновлення сигнатур, що є основою бізнес-моделі виробників антивірусів.

Своєчасна доставка нових сигнатур користувачів також є серйозною проблемою для виробників ПЗ. Сучасні віруси і хробаки розповсюджуються з такою швидкістю, що на момент випуску сигнатури та доставки її на комп'ютер користувачів епідемія вже може досягти своєї найвищої точки і охопити весь світ. За опублікованими даними доставка сигнатури займає від 11 до 97 годин в залежності від виробника, у той час, як теоретично вірус може захопити весь інтернет менше, ніж за 30 секунд. У більшості програм з безпеки база сигнатур є ядром продукту – найбільш трудомісткою і цінною частиною. Саме тому більшість вендорів воліє тримати свої сигнатури закритими – хоча й у цій галузі існує ряд відкритого ПЗ, а також дослідження щодо зворотної розробки закритих сигнатур.

До плюсів даного підходу можна віднести наступне:

- Можливість визначати конкретну атаку з високою точністю;
- Мала кількість хибних викликів.

До недоліків методу сигнатурного аналізу можна віднести:

- Беззахисність перед поліморфними вірусами та зміненими версіями того ж вірусу;
- Потреба у регулярному та вкрай оперативному оновленні;
- Потреба кропіткого ручного аналізу вірусів;
- Нездатність виявити будь-які нові атаки.

#### 1.4 Цифровий підпис

Цифровий підпис є математичною схемою для перевірки автентичності цифрових повідомлень або документів. Дійсний цифровий підпис, якщо дотримані передумови, дає одержувачу дуже високу впевненість у тому, що повідомлення було створено відомим відправником, і що повідомлення не було змінено під час передачі[4]. Цифрові підписи є стандартним елементом більшості наборів криптографічних протоколів і зазвичай використовуються для розповсюдження програмного забезпечення, фінансових транзакцій, програмного забезпечення для управління контрактами та в інших випадках, коли важливо виявити підробку або втручання.

Цифрові підписи використовують асиметричну криптографію. У багатьох випадках вони забезпечують рівень перевірки та безпеки повідомлень, надісланих через незахищений канал: належним чином реалізований цифровий підпис дає одержувачу підстави вважати, що повідомлення було надіслано заявленим відправником. Цифрові підписи багато в чому еквівалентні традиційним рукописним підписам, але правильно реалізовані цифрові підписи підробити складніше, ніж рукописні. Схеми формування цифрового підпису базуються на криптографії, і для їх ефективності необхідна правильна реалізація. Повідомлення з цифровим підписом можуть бути будь-якими, представленими у вигляді бітового

рядка: наприклад, електронна пошта, контракти або повідомлення, надіслані через якийсь інший криптографічний протокол.

Схема цифрового підпису зазвичай складається з трьох алгоритмів:

- Алгоритм генерації ключів, який вибирає приватний ключ рівномірно випадковим чином з набору можливих закритих ключів. Алгоритм виводить закритий ключ і відповідний відкритий ключ.
- Алгоритм підпису, який, надавши повідомлення та закритий ключ, створює підпис.
- Алгоритм перевірки підпису, який, враховуючи повідомлення, відкритий ключ і підпис, приймає або відхиляє заяву про автентичність повідомлення.

Необхідні дві основні властивості. По-перше, автентичність підпису, створеного з фіксованого повідомлення та фіксованого приватного ключа, можна перевірити за допомогою відповідного відкритого ключа. По-друге, генерувати дійсний підпис для сторони, не знаючи приватного ключа цієї сторони, має бути неможливо з обчислювальної точки зору. Цифровий підпис – це механізм автентифікації, який дає змогу творцю повідомлення прикріпити код, який діє як підпис. Алгоритм цифрового підпису (DSA), розроблений Національним інститутом стандартів і технологій, є одним із багатьох прикладів алгоритму підпису.

## 1.5 RSASSA-PSS

RSASSA-PSS (RSA Signature Scheme with Appendix-Probabilistic Signature Scheme)[5] – асиметричний алгоритм цифрового підпису. Заснований на принципі кодування PSS, запропонованому в 1996 році авторами Mihir Bellare і Phillip Rogaway. Внесений в стандарт PKCS#1 v2.1 від 2002 року, розроблений RSA Laboratories, США.

При використанні цього алгоритму цифрового підпису виконується наступна послідовність дій (Рис. 1.2):

1. PSS-кодування. На цьому етапі застосовуються функція кодування для підпису повідомлення;



2. RSA-підпис. Переводимо результат PSS-кодування у ціле число. Потім застосовується закритий ключ ( $s = m^d \bmod n$ ). Останнім етапом виконується перетворення числа у строку та повернення підпису;
3. Повернення підпису.

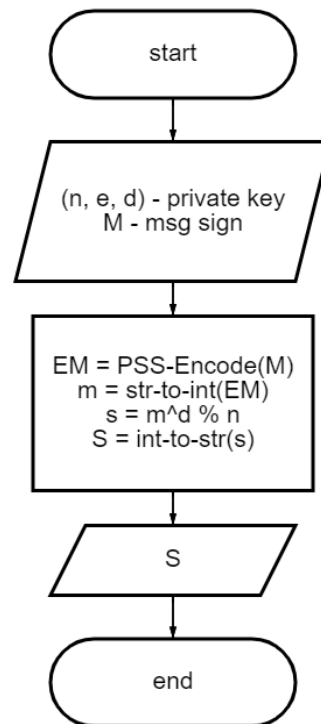


Рисунок 1.2 – Блок-схема алгоритму створення RSASSA-PSS підпису

Наступний важливий крок це розібратися у тому як працює PSS-Encode. Алгоритм описано далі (Рис. 1.3):

1. Якщо довжина  $M$  більша максимально можливої довжини вхідного рядка обраної хеш-функції ( $2^{61} - 1$  байт для SHA-1), повертається повідомлення «повідомлення дуже довге» і робота припиняється;
2.  $mHash = Hash(M)$ , тобто виконується хешування повідомлення;
3. Генерується випадковий рядок  $salt$  довжини  $sLen$ ; якщо  $sLen = 0$ ,  $salt$  – порожній рядок.
4.  $M' = (0x)00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 || mHash || salt$ , рядок довжини  $(8 + hLen + sLen)$ , перші 8 байт якої – нульові;

5.  $H = \text{Hash}(M')$ , рядок довжини  $hLen$ .
6.  $DB = PS || 0x01 || salt$ , рядок для генерації верхніх біт;
7.  $dbMask = \text{MGF}(H, emLen - hLen - 1)$ , рядок для створення верхніх біт підпису;
8.  $maskedDB = DB \oplus dbMask$ .
9. Старші біти встановлюються  $maskedDB$ ;
10.  $TF = 0xBC$ ;
11.  $EM = maskedDB || H || TF$  – загальна структура encode;
12. Повернення  $EM$ .

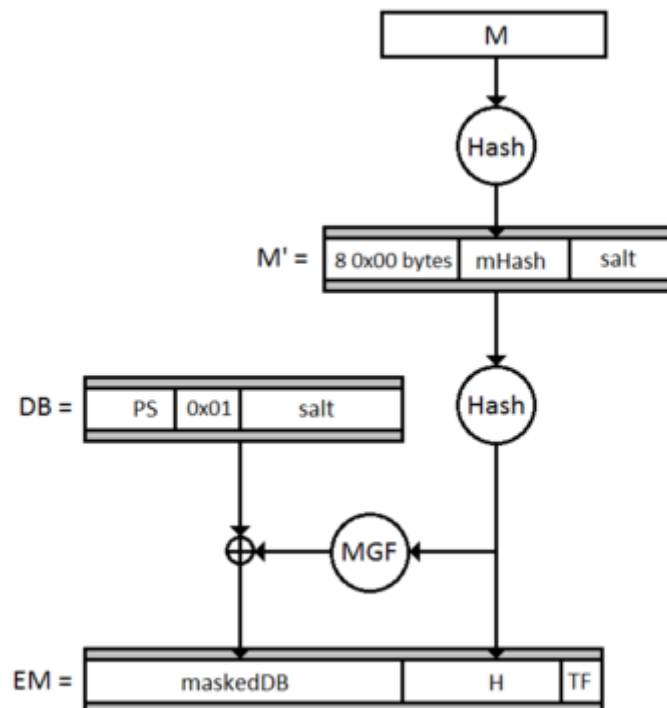


Рисунок 1.3 – Формування  $EM$

Перевірка підпису відбувається за дещо оберненим алгоритмом (Рис. 1.4).

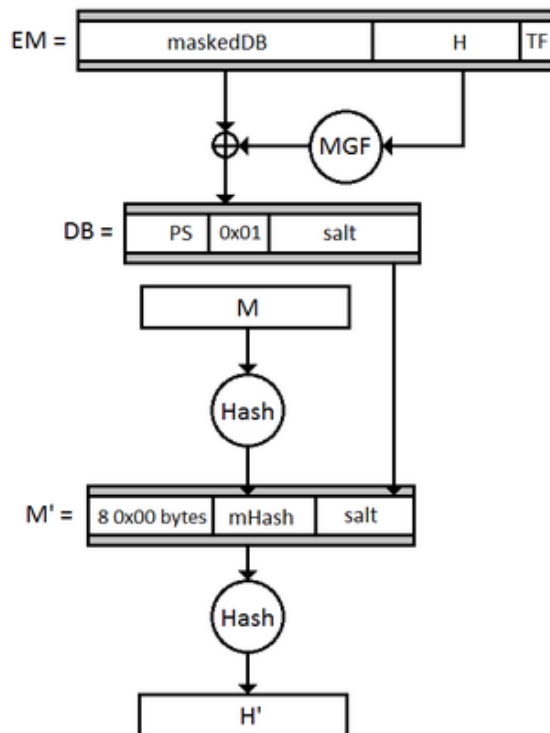


Рисунок 1.4 – Перевірка *EM*

Останнім за порядком, проте не за значенням, є функція генерації маски. З точки зору випадкових послідовностей вона цікавить нас більше за все. Опишемо MGF, яка використана в PSS-функціях. MGF приймає на вхід бінарний рядок довільної довжини і бажану довжину вихідного рядка й видає бінарний рядок бажаної довжини. На довжину вхідний і вихідний рядків можуть накладатися обмеження, але вони зазвичай дуже великі. MGF детермінована, вихідний рядок повністю визначається вхідним рядком. Вихідні дані MGF повинні бути псевдовипадковими, тобто знаючи частину вихідного рядка, має бути практично неможливо передбачити частину вихідного рядка. Ця властивість необхідна, так як на неї покладається доказ надійності алгоритму.

## 2 ФОРМАЛІЗАЦІЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

### 2.1 Математична постановка задачі

Фактично стоїть задачі реалізації універсальних алгоритмів генерації псевдовипадкових послідовностей за рахунок лінійного зсувного регістру зі зворотними зв'язками та матричного зсувного регістру.

Зсувний регістр з лінійною зворотною зв'язком (LFSR) – це регістр зсуву, вхідний біт якого є лінійною функцією його попереднього стану. Зазвичай використовується лінійна функцією XOR. Таким чином, LFSR найчастіше є регістром зсуву, вхідний біт якого керується операцією XOR деяких бітів загального значення регістра зсуву. Початкове значення LFSR називається початковим, і оскільки робота регістра є детермінованою, потік значень, вироблених регістром, повністю визначається його поточним станом. Аналогічно, оскільки регістр має скінченну кількість можливих станів, він в кінцевому підсумку повинен увійти в повторюваний цикл. Однак LFSR з добре вибраною функцією зворотного зв'язку може створити послідовність бітів, яка виглядає випадковою і має дуже довгий цикл.

Застосування LFSR включають генерацію псевдовипадкових чисел, псевдошумових послідовностей, швидких цифрових лічильників, тощо. Одним з плюсів LFSR є можливість як програмної так і апаратної реалізації. Математика циклічної перевірки надмірності, яка використовується для швидкої перевірки на помилки передачі, тісно пов'язана з такою в LFSR. Загалом, арифметика LFSR робить їх дуже елегантними як об'єкт для вивчення та впровадження. Можна створити відносно складну логіку з простих будівельних блоків. Однак слід розглянути й інші методи, менш елегантні, але кращі.

Слід окремо зазначити як формується  $M$ -послідовності за допомогою LFSR. Для формування  $M$ -послідовності, котра дорівнює  $N$  біт слід використовувати неспростований поліном  $N$ -го ступеню. Неспростований поліном це такий поліном, котрий неможливо представити лінійною комбінацією інших поліномів. Деякий список неспростованих поліномів можна побачити далі (Табл. 2.1).

Таблиця 2.1 – Деякі неспростовані поліноми

Кількість біт (ступень поліному)	Представлення поліному	Період ( $2^n - 1$ )
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^3 + 1$	255

Щоб правильно задати поліном потрібно отримати коефіцієнти від поліному (наприклад у якості двійкового слова, Рис. 2.1).

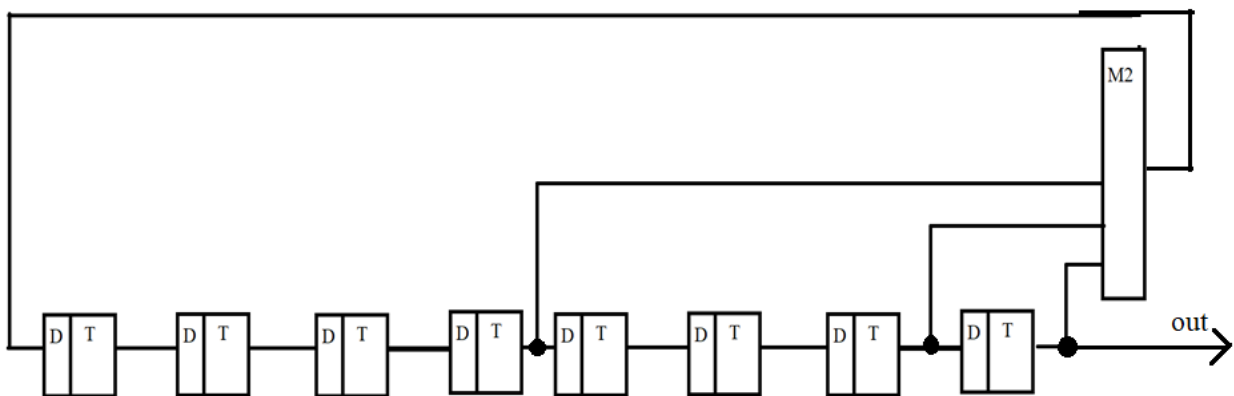


Рисунок 2.1 – LFSR для поліному  $x^8 + x^4 + x + 1$  (100010011)

Для простоти також використовують таблиці неспростованих поліномів (Рис. 2.2). Кожний поліном у цій таблиці задається у вигляді 8-річного числа. Для розшифрування цього поліному потрібно просто перевести число у двійковий формат та розписати коефіцієнти.

Степень 1					
Степень 2	1 7H				
Степень 3	1 13F				
Степень 4	1 23F	3 37D	5 07		
Степень 5	1 45E	3 75G	5 67H		
Степень 6	1 103F 11 155E	3 127B 21 007	5 147H	7 111A	9 015
Степень 7	1 211E 11 325G	3 217E 13 203F 19 313H	5 235E 21 345G	7 367H	9 277E
Степень 8	1 435E 11 747H 23 543F 31 037	3 567B 13 453F 15 727D 25 433B 85 007	5 763D 17 023 37 537F	7 551E 19 545E 43 703H	9 675C 21 613D 45 471A
Степень 9	1 1021E 11 1055E 23 1751E 39 1715E 55 1275E	3 1131E 13 1167F 25 1743H 41 1563H 73 0013	5 1461G 17 1333F 29 1553H 45 1175E 77 1511C	7 1231A 19 1605G 35 1401C 51 1725G 83 1425G	9 1425G 21 1027A 37 1157F 53 1225E 85 1267E
Степень 10	1 2011E 11 2065A 23 2033F 35 3023H 47 3177H 59 3471G 83 3623H 99 0067 147 2355A 179 3211G	3 2017B 13 2157F 25 2443F 37 3543F 49 3525G 69 2701A 85 2707E 101 2055E 149 3025G 341 0007	5 2415E 17 3515G 29 2461E 41 2745E 53 2617F 73 3507H 89 2327F 103 3575G 155 2251A	7 3771G 19 2773F 31 3043D 43 2431E 55 3453D 75 2437B 91 3265G 107 3171G 171 3315C	9 2257E 21 3753D 33 0075C 45 3061C 57 3121C 77 2413B 93 3777D 109 2047F 173 3337H

Рисунок 2.2 – Таблица неспростованих поліномів

Формувати числа цього реєстру можна не лише апаратно, але і програмно. Для цього можна використовувати математичний підхід. Одна з можливостей – системи розносних рівнянь. У загальному вигляді вона має такий вигляд:

$$\sum_{j=0}^k C_{k-j} \cdot y_{i-j} = 0, \text{ де } C_0 \cdot C_k = 1$$

Також існує матричний метод вирішення цієї задачі[6]. Проте спочатку слід поговорити про спеціальну форму представлення матриці коефіцієнтів. Якщо поліном задається як:

$$f(x) = C_n x^n + C_{n-1} x^{n-1} + \dots + C_1 x^1 + C_0 x^0$$

Тоді коефіцієнти цього поліному можна представити у вигляді:

$$F = \begin{pmatrix} C_{n-1} & C_{n-2} & \dots & C_1 & C_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

Тобто у такій формі перший рядок такої матриці дорівнює коефіцієнтам поліному (окрім старшого, оскільки  $C_n = 1$ ), а елементи під головною діагоналлю дорівнюють одиниці. Використовуючи таку форму коефіцієнтів можна отримати наступну матричну форму:

$$Y_{i+1} = F \cdot Y_i$$

$Y_i$  – вектор стані

$F$  – матриця коефіцієнтів

$Y_{i+1}$  – вектор станів наступного кроку

Також потрібно розробити відповідний алгоритм для матричного зсувного регістру. Матричний зсувний регістр це такий самий регістр, проте у матричному вигляді (Рис. 2.3).

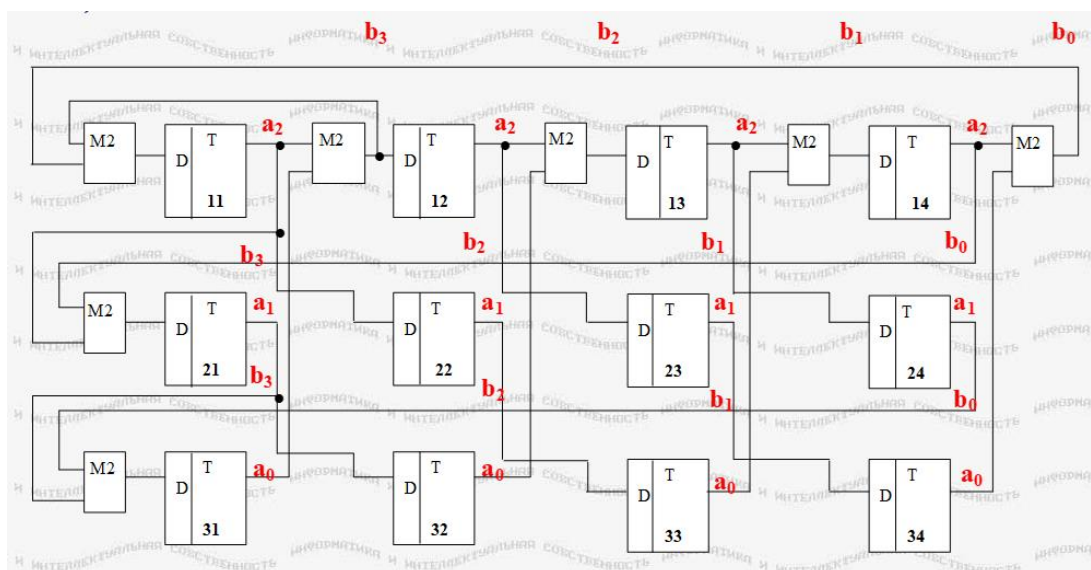


Рисунок 2.3 – Схема апаратної реалізації матричного зсувного регістру

Для програмної реалізації також має сенс використовувати матричну форму:

$$S_{t+1} = A \cdot S_t \cdot B$$

Проте слід зазначити, що матриця коефіцієнтів  $B$  має не такий самий вигляд, як матриця  $A$ . Загалом матриці  $A$  та  $B$  формуються так само, як матриця коефіцієнтів  $F$ . Матриця  $B$  є транспонованою відносно  $A$ . Для отримання матриці  $B$  потрібно:

$$F^T = B$$

## 2.2 Розробка архітектури та функціональності програмної системи

Для реалізації вищеописаних алгоритмів було використано мову програмування Java. Спочатку слід розробити відповідні алгоритми генерування елементів (Рис. 2.4 – 2.5).

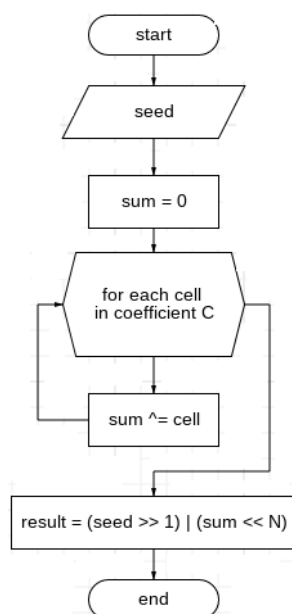


Рисунок 2.4 – Блок-схема алгоритму LFSR

Загалом блок-схему LFSR можна представити наступним псевдокодом:



```

int sum = (C & 1) & (Yk & 1);
for(int i = 1; i < n; i++) sum ^= ((C >> i & 1) & (Yk >> i & 1));
Yk = Yk >> 1;
Yk |= (sum << (n - 1));

```

Якщо ж використовувати математико-програмістичний синтаксис можна вивести таку формулу:

$$Y_k = (Y_{k-1} \gg 1) \vee \left( \sum_{n=0}^N C_n Y_{k-1,n} \wedge 1 \ll N \right)$$

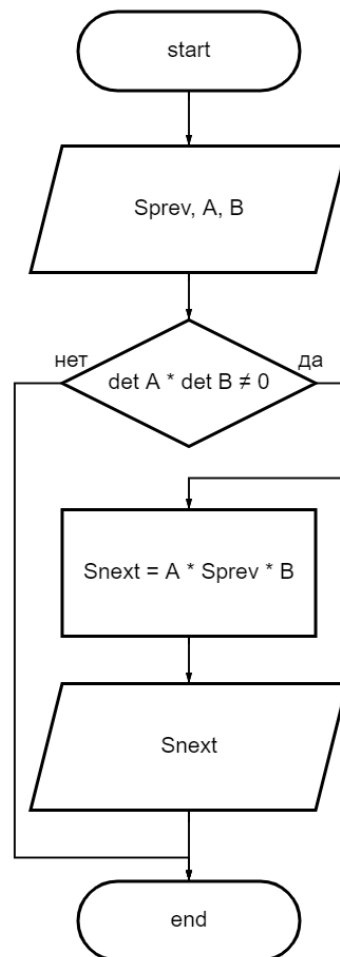


Рисунок 2.5 – Блок-схема MSR

Для реалізації матричного зсувного регістру було використано просто матричну формулу.

### 3 ПРОГРАМУВАННЯ ЗАСТОСУНКУ

#### 3.1 Обрані технології розробки

У якості основної технології було обрано мову програмування Java[7]. Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Обрано саме мову Java через те, що вона має потужні інструменти розробки інтерфейсу користувача (Abstract Window Toolkit[8] та Swing[9]), а також загальний об'єктно орієнтований підхід до програмування.

#### 3.2 Розробка LFSR

Для реалізацію лінійного зсувного регістру зі зворотними зв'язками було створено його об'єктно-орієнтовану абстракцію (Додаток А). Загальний алгоритм роботи з даним класом наступний:

1. Створення екземпляру класу зсувного регістру з параметрами  $n$  (кількість тригерів) та  $C$  (коефіцієнти регістру);
2. Отримання наступного стану залежно від заданого.

До додаткових можливостей цього класу належать отримання теоретичного періоду послідовності:

$$T_{th} = 2^n - 1$$

та реального періоду. Реальний період розраховується як кількість елементів послідовності, поки послідовність не починає повторюватись. Також є можливість отримати математичне очікування:

$$M_x = (-1) \cdot T_{th}^{-1} = -\frac{1}{2^n - 1}$$

А також дисперсію:

$$D_x = 1 - (M_x)^2 = 1 - \left(\frac{1}{2^n - 1}\right)^2$$

### 3.3 Розробка MSR

Для реалізації матричного зсувного регістру спочатку слід реалізувати додатковий матричний клас (Додаток Б). Особливістю даного класу буде можливість перемноження матриць (Рис. 3.1).

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix};$$

$$\alpha \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} \alpha a_{11} & \alpha a_{12} \\ \alpha a_{21} & \alpha a_{22} \end{pmatrix}.$$

Рисунок 3.1 – Перемноження матриць між собою та на скаляр

Також однією з можливостей розробленого класу матриці є можливість отримувати визначник матриці:

```
public double determinant() {
    if(N != M) throw new IllegalArgumentException("N != M");
    if(N == 1) return matrix[0][0];
    double res = 0.0;
    for(int i = 0; i < N; i++) {
        res += Math.pow(-1, 2 + i) * matrix[0][i] * cofactor(0, i).determinant();
    }
    return res;
}
```

Де *cofactor*:

```
public Matrix cofactor(int i, int j) {
    Matrix b = new Matrix(N - 1, M - 1);
    int pos1 = 0, pos2 = 0;
    for (int k = 0; k < N; k++) {
        if(k == i) continue;
        for (int l = 0; l < M; l++) {
            if (l != j) {
                b.matrix[pos1][pos2++] = matrix[k][l];
            }
        }
        pos1++;
    }
    return b;
}
```

```

    }
}
pos1++;
pos2 = 0;
}
return b;
}

```

Алгоритм підрахунку визначника матриці можна записати у вигляді розкладання мінору за рядком:

$$\Delta = \sum_{j=1}^n (-1)^{1+j} a_{1j} M_j^{-1}$$

$M_j^{-1}$  – додатковий міно́р матриці.

Останньою особливістю розробленого класу матриці є можливість отримання матриці коефіцієнтів з самих коефіцієнтів поліному:

```

public static Matrix fromCoefficient(int n, int C) {
    Matrix F = new Matrix(n, n);
    for(int shift = 0; shift < n; shift++) {
        F.matrix[0][shift] = (double) (C >> (n - shift - 1) & 1);
        if(shift == 0) continue;
        F.matrix[shift][shift - 1] = 1.;
    }
    return F;
}

```

Відповідний клас матричного зсувного регістру (Додаток В) дуже простий.

Фактично нас цікавить лише формування наступного елементу:

```

public Matrix next(Matrix state) {
    if(state.getN() != A.getN() || state.getM() != B.getN()) throw new
    IllegalArgumentException("state should be (" + A.getN() + ", " + B.getN() + ")");
    Matrix beforeSum = A.mul(state).mul(B);
    beforeSum.forEach(v -> (v % 2));
    return beforeSum;
}

```

Фактично відбувається застосування матричної формули. Також можна отримати теоретичний та практичний періоди цього регістру.

Теоретичний період розраховується як:

$$T_{th} = \text{gcd}(T_a, T_b) = T_a \cdot T_b$$

Практичний період розраховується як кількість можливих станів доти, доки стан не повторюється з початковим.

### 3.4 Розробка інтерфейсу користувача

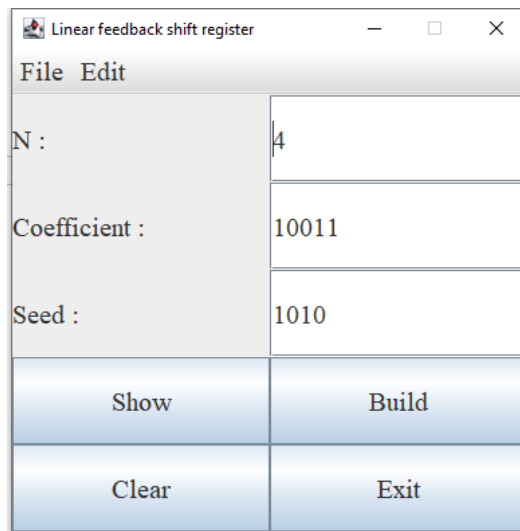
Для демонстрації роботи розробленого програмного продукту також реалізовано графічний інтерфейс користувача (Graphical User Interface, GUI). Увесь проект поділено на відповідні класи фрейми, котрі відповідають за окремі можливості роботи з регістрами: CustomPolynomialFrame, FrameBase, IrreduciblePolynomialFrame, LFSRBuildFrame, LFSRShowFrame, LFSRFrame, MSRFrame, MSRShowFrame, MSRSetSeedFrame.

Таким чином у користувача повинна бути можливість працювати як з лінійним зсувним регістром так і матричним зсувним регістром. Також у користувача повинна бути можливість обрати поліноми у якості початкових станів, а також задавати власні початкові стани.

## 4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОДУКТУ

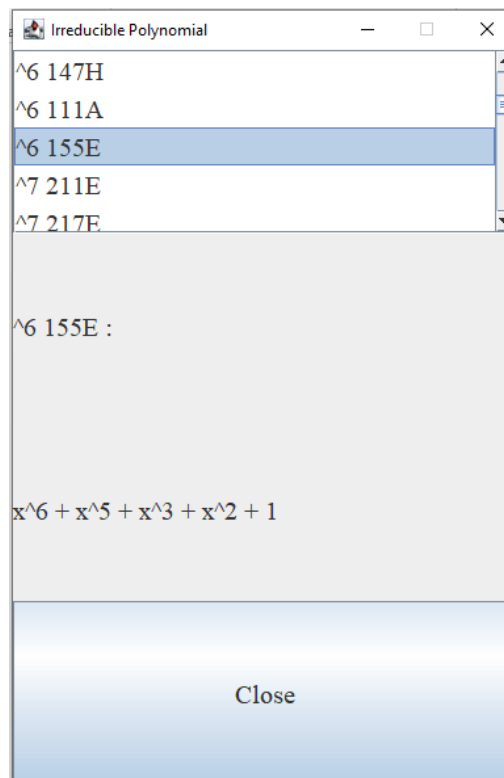
### 4.1 Тестування LFSR

При старті LFSR (Рис. 4.1) користувач може одразу задати власні стани, а також обрати прості поліноми (Рис. 4.2).



The screenshot shows a window titled "Linear feedback shift register". It has a menu bar with "File" and "Edit". Below the menu bar, there are three input fields: "N:" with the value "4", "Coefficient:" with the value "10011", and "Seed:" with the value "1010". At the bottom, there are four buttons arranged in a 2x2 grid: "Show", "Build", "Clear", and "Exit".

Рисунок 4.1 – Головна сторінка LFSR



The screenshot shows a window titled "Irreducible Polynomial". It has a list box containing several polynomial entries: " $x^6 147H$ ", " $x^6 111A$ ", " $x^6 155E$ ", " $x^7 211E$ ", and " $x^7 217E$ ". The entry " $x^6 155E$ " is selected and highlighted in blue. Below the list box, there is a text area displaying the polynomial " $x^6 155E :$ " followed by the expression " $x^6 + x^5 + x^3 + x^2 + 1$ ". At the bottom, there is a "Close" button.

Рисунок 4.2 – Вибір простих поліномів

Надалі у користувача є можливість побудувати цей регістр (Рис. 4.3), а також зберігати схему регістру з станом (Рис. 4.4).

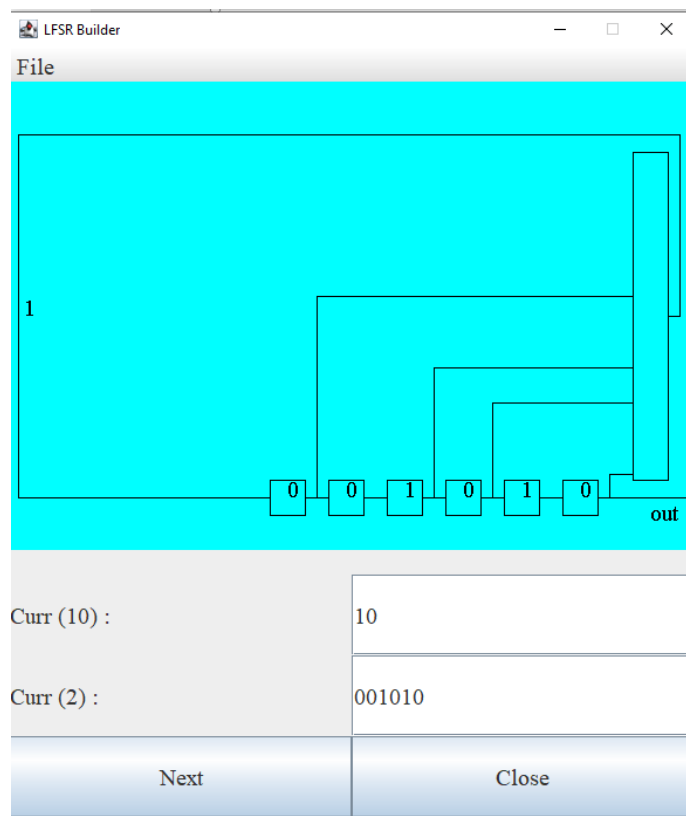


Рисунок 4.3 – Побудова схеми LFSR

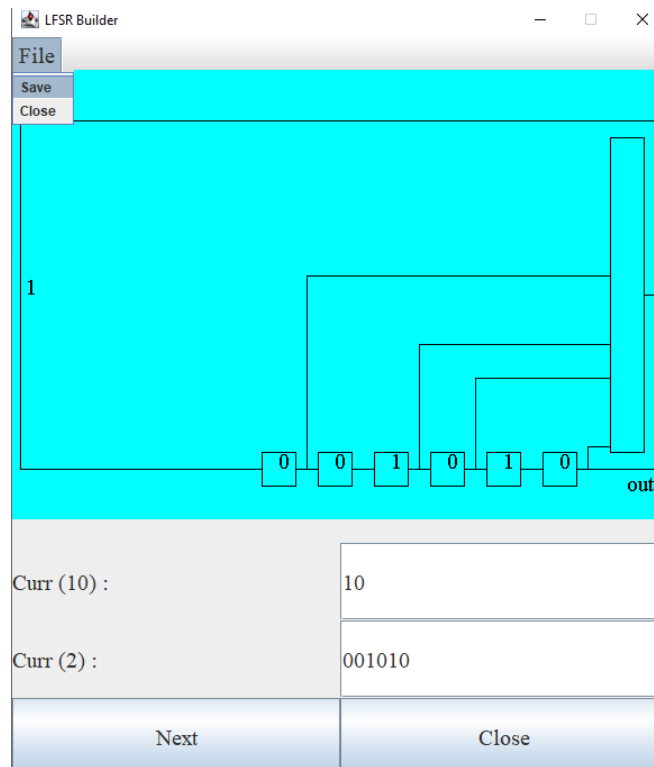


Рисунок 4.4 – Збереження схеми як зображення

Також користувач має можливість отримати повний період послідовності та деякі характеристики регістру (Рис. 4.5), а також зберігати період регістру (Рис. 4.6).

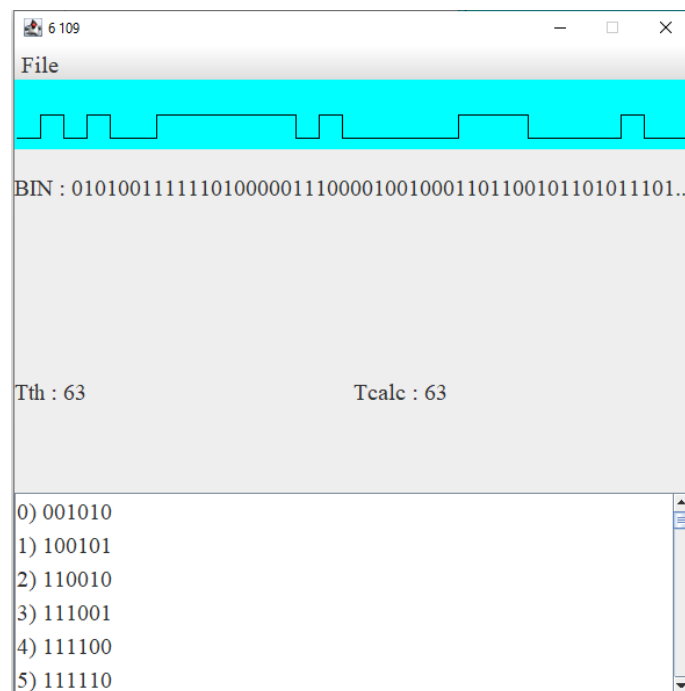


Рисунок 4.5 – Отримання повного періоду регістру



```
BIN : 0101111000100110  
Tth : 15  
Tcalc : 15  
0) 1010  
1) 1101  
2) 1110  
3) 1111  
4) 0111  
5) 0011  
6) 0001  
7) 1000  
8) 0100  
9) 0010  
10) 1001  
11) 1100  
12) 0110  
13) 1011  
14) 0101  
15) 1010
```

Рисунок 4.6 – Збереження періоду регістру у файл

## 4.2 Тестування MSR

При запуску MSRFrame (Рис. 4.7) користувач може характеристики регістру та початковий стан (Рис. 4.8).

The image shows a graphical user interface for the MSRFrame application. It features several input fields and buttons. The fields are labeled 'N:', 'Ca:', 'M:', 'Cb:', and 'Seed:'. The 'N:' field contains the value '3', 'Ca:' contains '1101', 'M:' contains '4', and 'Cb:' contains '11001'. The 'Seed:' field is followed by a 'SetSeed' button. At the bottom of the interface, there are two buttons: 'Show' and 'Exit'.

Рисунок 4.7 – Початковий екран MSR

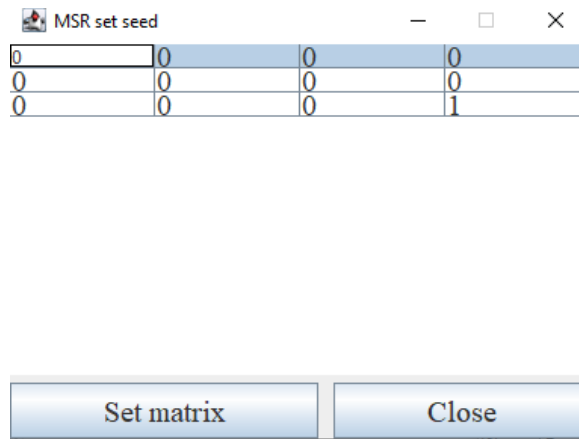


Рисунок 4.8 – Можливість задати початковий стан регістру

Далі у користувача є можливість промодельовати цей регістр (Рис. 4.9). Є можливість як ручного моделювання (отримання стану за станом) так і автоматичного моделювання. Користувач може отримати деякі характеристики прямо на цьому екрані, а також отримати додаткові характеристик послідовності, яка утворена виділенням (правим нижнім) елементом регістру (Рис. 4.10).

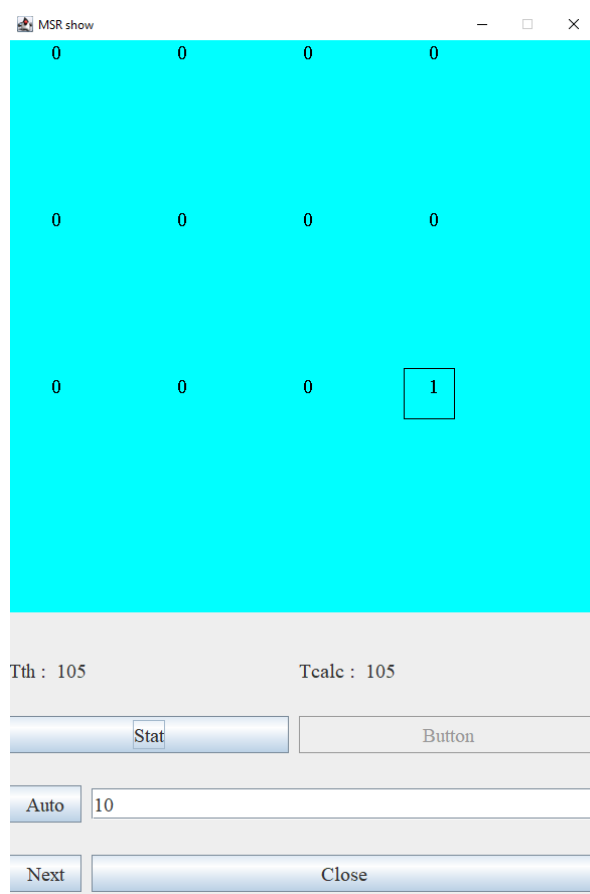


Рисунок 4.9 – Моделювання регістру

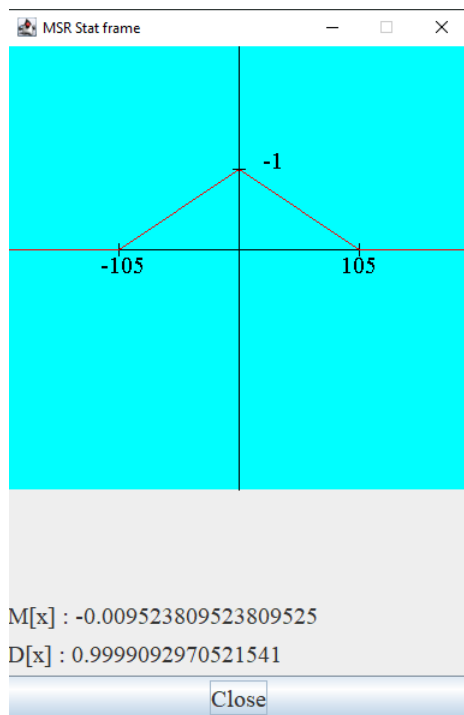


Рисунок 4.10 – Характеристики послідовності

Також у користувача є можливість отримати функцію автокореляції з отриманим рядом на  $\pm 1$ . Слід зазначити, що загальна функція автокореляції бінарного рядку має загальний вигляд (Рис. 4.11) та формується як:

$$R_x(\tau) = \begin{cases} -\frac{1}{2^n - 1}, \tau \neq 0 \\ 1, \tau = 0 \end{cases}$$

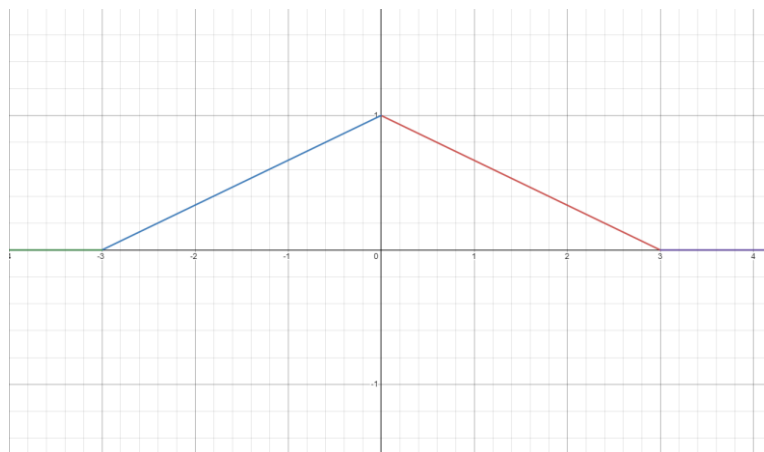


Рисунок 4.11 – Загальний вигляд функції автокореляції

## ВИСНОВКИ

В процесі виконання курсової роботи досліджено та представлено теоретичні відомості що стосуються проблем пов'язаних з випадковими числами. Також освітлено поняття псевдовипадкових бінарних послідовностей та сфери їх застосування у сучасних комп'ютерних технологіях.

Під час виконання лабораторної роботи було створено програмний додаток, котрий моделює роботу лінійних зсувних регістрів зі зворотними зв'язками та матричні зсувні регістри. Розроблений програмний додаток має лаконічний та інтуїтивно зрозумілий інтерфейс користувача, що дозволяє не розбиратися у складній структурі програми, а одразу перейти до моделювання.

До додаткових можливостей розробленого додатку можна також віднести можливість використовувати неспростовані поліноми, котрі вже задані у системі, а також створювати власні таблиці поліномів для збереження при подальшому використанні.

Ще однією важливою можливістю розробленого додатку є можливість отримувати статистики генераторів такі як математичне очікування, дисперсія, а також теоретичний та практичний періоди. У користувача є можливість побудови автокореляційної функції бінарних послідовностей.

У результаті виконання роботи було проведено як теоретичне так і практичне дослідження генераторів псевдовипадкових чисел на прикладі лінійних зсувних регістрів зі зворотними зв'язками та матричних зсувних регістрів.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Генерація випадкових чисел, котрі можна довіряти. [Електронний ресурс] – <https://www.random.org/>
2. L'Ecuyer, Pierre. Random Number Generation // Springer Handbooks of Computational Statistics : Глава. — 2007. — С. 93—137. — doi:10.1002/9780470172445.ch4.
3. McEliece R. J. Finite Field for Scientists and Engineers, Kluwer Academic Publishers, 1987.
4. Paul, Eliza (12 September 2017). "What is Digital Signature – How it works, Benefits, Objectives, Concept". EMP Trust HR.
5. Bellare, Mihir; Rogaway, Phillip. "PSS: Provably Secure Encoding Method for Digital Signatures" 2017-08-10.
6. Klein, A. (2013). "Linear Feedback Shift Registers". Stream Ciphers. London: Springer. pp. 17–18. doi:10.1007/978-1-4471-5079-4\_2. ISBN 978-1-4471-5079-4.
7. Офіційний сайт мови програмування Java. <https://www.java.com/en/>
8. Fowler, Amy (1994). "Mixing heavy and light components". Sun Microsystems. Archived from the original on 23 December 2011. Retrieved 17 December 2008.
9. Yap, Chee (2003-04-30). "JAVA SWING GUI TUTORIAL". New York University (NYU). Retrieved 2018-11-03.

## Додаток А

```
package logic;

/**
 * Basic linear feedback shift register
 * Uses no additional memory
 */
public class LFSR {

    /**
     * Initial state
     */
    private final int C;

    /**
     * Count of triggers
     */
    private final int n;

    /**
     * Constructor for linear-feedback shift register
     * with {@code n} triggers and {@code C} state
     *
     * @param n count of triggers
     * @param C coefficient for register
     */
    public LFSR(int n, int C) {
        if(n <= 0 || n > 32) throw new IllegalArgumentException("n <= 0 || n > 32");
        if(((C & (1 << n)) == 0) || ((C & 1) == 0)) throw new
        IllegalArgumentException("C0 * Cn != 1");
        //(C & 1) == 0 ||
        this.n = n;
        this.C = C;
    }

    /**
     * Method generate result of sum (XOR) of current state and {@code num}
     *
     * @param num current register state
     * @return XOR by coefficients
     */
    public int outputOfSum(int num) {
        int sum = (C & 1) & (num & 1);
        for(int i = 1; i < n; i++) sum ^= ((C >> i & 1) & (num >> i & 1));
        return sum;
    }

    /**
     * Method to generate next value from current state
     *
     * @param num current state of register
     * @return next state of register
     */
    public int next(int num) {
        if(num == 0) return 0;
        else if(num < 0) throw new IllegalArgumentException();
        int sum = outputOfSum(num);
        num = num >> 1;
        num |= (sum << (n - 1));
        return num;
    }
}
```

```

/**
 * @return coefficient
 */
public int getC() {
    return C;
}

/**
 * @return count of triggers
 */
public int getN() {
    return n;
}

/**
 * @return theoretical period of linear feedback shift register
 */
public int getT() { return (int) (Math.pow(2, n) - 1); }

/**
 * @return actual period of linear feedback shift register
 */
public int getTActual() {
    int T = 1;
    int curr = next(1);
    while(curr != 1) {
        curr = next(curr);
        T++;
    }
    return T;
}

/**
 * @return expected value of binary sequence
 */
public double getMx() {
    return - 1. / getT();
}

/**
 * @return dispersion of binary sequence
 */
public double getDx() {
    return 1. - Math.pow(getMx(), 2);
}

/**
 * Method to get basic integer iterable from curr {@code seed}
 *
 * @param seed basic state of triggers
 * @return {@code new LFSRIterable} to iterate through collection
 */
public Iterable<Integer> getIterable(int seed) {
    return new LFSRIterable(this, seed);
}
}

```

## Додаток Б

```
package logic;

import java.util.Arrays;
import java.util.Objects;
import java.util.function.DoubleConsumer;
import java.util.function.DoubleFunction;
import java.util.function.DoubleSupplier;

public class Matrix {

    private Double[][] matrix;
    private final int N, M;

    private void checkSize(Matrix m2) {
        if(N != m2.N || M != m2.M) throw new IllegalArgumentException("m2 should
have same size. Expected (" + N + ", " + M + ") got (" + m2.N + ", " + m2.M + ").");
    }

    private void checkSizeMul(Matrix m2) {
        if(M != m2.N) throw new IllegalArgumentException("(N, M) * (M, K) = (N,
K)");
    }

    public static double mulVector(Double[] v1, Double[] v2) {
        if(v1 == null || v2 == null || v1.length != v2.length) throw new
IllegalArgumentException("v1.length != v2.length");
        double d = 0.0;
        for(int i = 0; i < v1.length; i++) {
            d += v1[i] * v2[i];
        }
        return d;
    }

    public static Matrix identity(int n) {
        Matrix m = new Matrix(n, n);
        for(int i = 0; i < n; i++) {
            m.matrix[i][i] = 1.;
        }
        return m;
    }

    public static Matrix fromCoefficient(int n, int C) {
        Matrix F = new Matrix(n, n);
        for(int shift = 0; shift < n; shift++) {
            F.matrix[0][shift] = (double) (C >> (n - shift - 1) & 1);
            if(shift == 0) continue;
            F.matrix[shift][shift - 1] = 1.;
        }
        return F;
    }

    public static Matrix ones(int n, int m) {
        Matrix a = new Matrix(n, m);
        a.forEach(() -> 1.);
        return a;
    }

    public Matrix(int n, int m) {
        N = n;
        M = m;
    }
}
```



```

        matrix = new Double[N][M];
        forEach(() -> 0.0);
    }

    public Matrix(Double[][] mat) {
        N = mat.length;
        M = mat[0].length;
        matrix = new Double[N][M];
        for(int i = 0; i < mat.length; i++) {
            System.arraycopy(mat[i], 0, matrix[i], 0, mat[i].length);
        }
    }

    public void set(int i, int j, Double val) {
        matrix[i][j] = val;
    }

    public Double get(int i, int j) {
        return matrix[i][j];
    }

    public Double[] getRow(int i) {
        return matrix[i];
    }

    public Double[] getColumn(int j) {
        Double[] column = new Double[N];
        for(int i = 0; i < N; i++) {
            column[i] = matrix[i][j];
        }
        return column;
    }

    public void setRow(int i, Double[] row) {
        System.arraycopy(row, 0, matrix[i], 0, row.length);
    }

    public void setColumn(int j, Double[] column) {
        for(int i = 0; i < column.length; i++) {
            matrix[i][j] = column[i];
        }
    }

    public int getN() {return N;}

    public int getM() {return M;}

    public Matrix add(Matrix m2) {
        checkSize(m2);
        Matrix C = new Matrix(N, M);
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < M; j++) {
                C.matrix[i][j] = matrix[i][j] + m2.matrix[i][j];
            }
        }
        return C;
    }

    public Matrix sub(Matrix m2) {
        return add(m2.mulM1());
    }

    public Matrix mul(Matrix m2) {

```

```

        checkSizeMul(m2);
        Matrix c = new Matrix(N, m2.M);
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < m2.M; j++) {
                c.matrix[i][j] = mulVector(getRow(i), m2.getColumn(j));
            }
        }
        return c;
    }

    public Matrix mulM1() {
        return mul(-1.);
    }

    public Matrix mul(Double scalar) {
        Matrix B = new Matrix(N, M);
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < M; j++) {
                B.matrix[i][j] = matrix[i][j] * scalar;
            }
        }
        return B;
    }

    public Matrix pow(Integer power) {
        if(power <= 0) throw new IllegalArgumentException("power <= 0");
        Matrix m = copy();
        for(int i = 0; i < power; i++) {
            m = m.mul(this);
        }
        return m;
    }

    public Matrix T() {
        Matrix t = new Matrix(M, N);
        for(int i = 0; i < N; i++) {
            t.setColumn(i, getRow(i));
        }
        return t;
    }

    public Matrix copy() {
        Matrix b = new Matrix(N, M);
        for(int i = 0; i < N; i++) {
            System.arraycopy(matrix[i], 0, b.matrix[i], 0, M);
        }
        return b;
    }

    public Matrix round(int precision) {
        Matrix b = copy();
        double p = Math.pow(10, precision);
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < M; j++) {
                b.matrix[i][j] = Math.round(matrix[i][j] * p) / p;
            }
        }
        return b;
    }

    public Matrix round() {
        return round(0);
    }

```

```

public double determinant() {
    if(N != M) throw new IllegalArgumentException("N != M");
    if(N == 1) return matrix[0][0];
    double res = 0.0;
    for(int i = 0; i < N; i++) {
        res += Math.pow(-1, 2 + i) * matrix[0][i] * cofactor(0, i).determinant();
    }
    return res;
}

public Matrix cofactor(int i, int j) {
    Matrix b = new Matrix(N - 1, M - 1);
    int pos1 = 0, pos2 = 0;
    for (int k = 0; k < N; k++) {
        if(k == i) continue;
        for (int l = 0; l < M; l++) {
            if (l != j) {
                b.matrix[pos1][pos2++] = matrix[k][l];
            }
        }
        pos1++;
        pos2 = 0;
    }
    return b;
}

public void forEach(DoubleConsumer consumer) {
    for(int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            consumer.accept(matrix[i][j]);
        }
    }
}

public void forEach(DoubleSupplier supplier) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < M; j++) {
            matrix[i][j] = supplier.getAsDouble();
        }
    }
}

public void forEach(DoubleFunction<Double> function) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < M; j++) {
            matrix[i][j] = function.apply(matrix[i][j]);
        }
    }
}

@Override
public String toString() {
    StringBuilder stringBuilder = new StringBuilder();
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < M; j++) {
            stringBuilder.append(matrix[i][j]).append(" ");
        }
        stringBuilder.append("\n");
    }
    return stringBuilder.toString();
}

@Override
public boolean equals(Object o) {

```

```
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Matrix matrix1 = (Matrix) o;
        return N == matrix1.N && M == matrix1.M && Arrays.deepEquals(matrix,
matrix1.matrix);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(N, M);
        result = 31 * result + Arrays.deepHashCode(matrix);
        return result;
    }
}
```

## Додаток В

```
package logic;

// Matrix shift register
public class MSR {

    private Matrix A, B;

    public MSR(Matrix A, Matrix B) {
        if(A.getN() != A.getM()) throw new IllegalArgumentException("A.N != A.M");
        if(B.getN() != B.getM()) throw new IllegalArgumentException("B.N != B.M");
        if(A.determinant() == 0 || B.determinant() == 0) throw new
        IllegalArgumentException("A & B should be non-singular");

        this.A = A;
        this.B = B;
    }

    public MSR(int N, int M, int Ca, int Cb) {
        this(Matrix.fromCoefficient(N, Ca), Matrix.fromCoefficient(M, Cb).T());
    }

    public Matrix next(Matrix state) {
        if(state.getN() != A.getN() || state.getM() != B.getN()) throw new
        IllegalArgumentException("state should be (" + A.getN() + ", " + B.getN() + ")");
        Matrix beforeSum = A.mul(state).mul(B);
        beforeSum.forEach(v -> (v % 2));
        return beforeSum;
    }

    public int getT() {
        int Ta = (int) (Math.pow(2, A.getN()) - 1);
        int Tb = (int) (Math.pow(2, B.getM()) - 1);

        /*int ta = Ta, tb = Tb;
        while(tb != 0) {
            int tmp = ta;
            ta = tb;
            tb = tmp % tb;
        }*/

        return Ta * Tb;
    }

    public int getTActual() {
        Matrix seed = Matrix.ones(A.getN(), B.getM());
        Matrix curr = seed.copy();
        curr = next(curr);
        int T = 1;
        while(!curr.equals(seed)) {
            curr = next(curr);
            T++;
        }
        return T;
    }

    /**
     * @return expected value of binary sequence
     */
    public double getMx() {
        return - 1. / getT();
    }
}
```

```
}

/**
 * @return dispersion of binary sequence
 */
public double getDx() {
    return 1. - Math.pow(getMx(), 2);
}

}
```