**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

Навчально–науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

# Пояснювальна записка

до магістерської роботи
на ступень вищої освіти магістр

на тему «**ЗАСТОСУВАННЯ МЕТОДУ ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ОСНОВІ НЕСТАНДАРТНИХ ВХІДНИХ ДАНИХ»**

Виконав: студент 6 курсу, групи ПДМ - 61
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Візер Антоній Миколайович
(прізвище та ініціали)
Керівник Трінтіна А. М.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

КИЇВ – 2022

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**Навчально–науковий інститут Інформаційних технологій**

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти «Магістр»
Спеціальність підготовки 121 Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного
забезпечення
_____О.В.Негоденко
“____” _____ 2022 року

# З А В Д А Н Н Я
## НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Візер Антоній Миколайович
(прізвище, ім’я, по батькові)

1.Тема роботи: «Застосування методу генерації псевдовипадкових чисел на основі нестандартних вхідних даних»

Керівник роботи Трінтіна Наталья Альбертівна, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від « 12 » жовтня 2022 року № 122

2. Строк подання студентом роботи 31.12.2022

3. Вихідні дані до роботи: Матеріали переддипломної практики, алгоритми генерації випадкових чисел, Тестування випадковості.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
4.1 Огляд предметної області
4.2 Модифікація алгоритму
4.3 Розробка програмного забезпечення
4.4 Проведення тестування та аналіз отриманих результатів

5. Перелік графічного матеріалу (презентація)
5.1 Мета, об’єкта та предмет дослідження
5.2 Аналіз предметної області
5.3 Модифікація алгоритму
5.4 Тестування
5.5 Аналіз отриманих результатів

6. Дата видачі завдання «14» жовтня 2022року

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської Роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на магістерську роботу | 14.10.2022 | |
| 2 | Огляд предметної області | 17.10.2022 | |
| 3 | Аналіз існуючих алгоритмів | 20.11.2022 | |
| 4 | Модифікація алгоритму | 22.11.2022 | |
| 5 | Розробка програмного бібліотеки | 23.11.2022 | |
| 6 | Тестування програмного бібліотеки | 24.11.2022 | |
| 7 | Аналіз отриманих результатів | 26.11.2022 | |
| 8 | Написання та оформлення пояснювальної записки | 27.11.2022 | |
| 9 | Розробка графічних та презентаційних матеріалів | 11.12.2022 | |
| 10 | Захист магістерської роботи | 17.01.2023 | |
| | | | |
| | | | |
| | | | |

Студент _____ Візер А. М..
( підпис ) (прізвище та ініціали)

Керівник роботи _____ Трінтіна Н. А.
( підпис ) (прізвище та ініціали)

# РЕФЕРАТ

Текстова частина магістерської роботи 70 с., 15 рис., 20 джерел.

Ключові слова: Випадковість, Випадкові числа, Генератори випадкових чисел, Генератор псевдовипадкових чисел.

*Об'єкт дослідження* – Генерація випадкових величин.

*Предмет дослідження* – Алгоритми генерації випадкових чисел.

*Мета роботи* – модифікація алгоритму для збільшення кількості можливих галузей застосування..

*Методи дослідження* – методи системного аналізу; аналіз наукової літератури; спостереження; абстрагування; узагальнення.

У роботі проведено аналіз генераторів псевдовипадкових виявлені галузі їх застосування. Проаналізовані вимоги до генераторів випадкових чисел проведено порівняльний аналіз існуючих генераторів.

Модифіковано алгоритм генерації випадкових чисел. Модифікований алгоритм пройшов перевірку статистичними тестами. Розроблено бібліотеку для генерації випадкових чисел. Також у ході роботи було розроблено програму для демонстрації роботи алгоритму генерації випадкових чисел на ослонові випадкових даних

Проведено аналіз властивостей створеного алгоритму та проведено порівняння з алгоритмом до модифікації.

# ЗМІСТ

ВСТУП ..... 10
1. АНАЛІЗ ГЕНЕРАЦІЇ ВИПАДНОВИХ ДАНИХ У СУЧАСНОМУ СВІТІ ..... 12
1.1 Визначення та використання випадковості ..... 12
1.2 Історія появи та застосування випадкових чисел у комп'ютерній техніці ..... 19
1.2.1 Перша машина генерації випадкових чисел ..... 20
1.2.2 ERNIE ..... 20
1.2.3 Перший персональний комп'ютер с генераторам випадковості. . 21
1.2.4 Поява та розвиток генераторів псевдо випадкових чисел ..... 22
1.2.5 Розвиток апаратних генераторів випадкових чисел ..... 23
1.3 Апаратні генератори випадкових чисел ..... 23
1.3.1 Фізичні явища що використовують для генерації випадкових чисел ..... 24
1.3.2 Використання видимих подій ..... 29
1.3.3 Проблеми та недоліки апаратних генераторів ..... 29
1.4 Програмні генератори випадкових чисел ..... 30
1.4.1Основні методи генерації випадкових чисел ..... 31
1.5 Використання генераторів випадкових чисел та вимоги до них ..... 34
1.5.1 Використання генераторів випадкових чисел у галузі криптографії ..... 34
1.5.2 Генератори випадкових чисел у імітаційному моделюванні ..... 36
1.5.3 Генератори випадкових чисел у розробці ігор ..... 36
1.5.4 ГПВЧ в галузі вимірювальної техніки ..... 38
2 ВИБІР АЛГОРИТМУ ТА ЙОГО МОДИФІКАЦІЯ ..... 39
2.1 Вибір алгоритму генерації випадкових чисел ..... 39
2.1.1 Вибір критеріїв для порівняльного аналізу. ..... 39
2.1.2 Аналіз Алгоритмів для порівння ..... 40
2.2 Аналіз даних які необхідні для обраного генератора. ..... 45
2.3 Вибір даних на основі яких будуть генеруватися випадкові числа ..... 49
2.4 Розробка алгоритму перетворення потоку даних у параметри алгоритму ..... 49
2.5 Принцип роботи модифікованого алгоритму генерації випадкових чисел ..... 51

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ..... 53
3.1 Створення бібліотеки для генерації чисел ..... 53
3.1.1 Обґрунтування вибору мови програмування. ..... 53
3.1.2 Структура бібліотеки ..... 54
3.1.2.1 Клас RandomGenerator. ..... 54
3.1.2.1 Клас PCG ..... 56
3.2 Перевірка статистичних властивостей алгоритму ..... 57
3.3 Генерація випадкових чисел на основі нестандартних даних ..... 59
3.4 Порівняння характеристик алгоритму при різних даних для генерації ..... 61

# УМОВНІ ПОЗНАЧЕННЯ

ПВГЧ – Псевдо випадковий генератор чисел.

АГЧ – Апаратний генератор чисел.

КБГВЧ – Криптографічно безпечний генератор випадкових чисел.

ЛКГ – Лінійний конгруентний генератор.

PCG (permuted congruential generator) – Переставлений конгруентний генератор.

LFSR (linear-feedback shift register) – Регістр зсуву з лінійним зворотним зв'язком.

# ВСТУП

**Актуальність дослідження.** В багатьох галузях на сьогоднішній день застосовують послідовності випадкових чисел. Най більш потрібні випадкові числа у галузі криптографії. Зазвичай в галузі криптографії використовують випадкові числа для генерації ключів для шифрування та дешифрування повідомлень. При цьому генерація випадкових чисел знаходиться на найнижчому рівні тому якщо послідовність легко вгадати то всі інші рівні захисту не матимуть сенсу. Тому для генераторів ц цій галузі найбільш жорсткі вимоги. Крім галузі криптографії генератори випадкових чисел використовують в різних дослідженнях при імітаційному моделюванні. Також генератори використовують при тестуванні різних пристроїв а також нейронних мереж. Ще генератори випадкових чисел застосовують підпас розробки комп'ютерних ігор.

Але комп'ютер це детермінована система яка не здатна згенерувати дійсно випадкову послідовність. Тому існують алгоритми які генерують послідовності чисел яку в тій чи іншій мірі наближені до випадкового розподілу. Але зазвичай ці алгоритми використовують якісь дані в якості початкових параметрів на основі яких і генеруються послідовність чисел яка схожа на випадкову.

Для забезпечення дійсно випадкових послідовностей зазвичай використовують апаратні пристрої які отримують дані з якихось складних фізичних процесів які у подальшому використовують у якості параметрів алгоритмів. Використання таких пристроїв потребує певних затрат тому воно не завжди можливе. Тому у якості параметрів алгоритмів використовують різні дані наприклад час дані про сам комп'ютер тощо. Але використання таких даних призводить до того що отримані числа легко вгадати. Через це для генерації випадкових чисел постійно шукають нові види даних які можна застосувати при генерації випадкових чисел.

**Мета дослідження –** створення алгоритму генерації випадкових чисел який для генерації випадкових чисел використовує не стандартні дані.

**Завдання дослідження:**

1. Дослідити сучасні алгоритми генерації випадкових чисел;
2. Визначати параметри алгоритмів та її можливості;
3. Визначити які дані можна використовувати для генерації випадкових чисел.
4. Модифікувати алгоритм для генерації випадкових чисел за допомогою не стандартних даних.

**Об’єкт дослідження -** Генерація випадкових величин.

**Предмет дослідження –** Алгоритми генерації випадкових чисел

**Методи дослідження:** методи системного аналізу; аналіз наукової літератури; спостереження; абстрагування; узагальнення.

**Наукова новизна одержаних результатів.** Результати дослідження пропонують спосіб генерації випадкових чисел за допомогою даних які раніше не використовувались при генерації випадкових чисел.

**Практичне значення одержаних результатів** полягає в тому, що на основі результатів цього дослідження було створено бібліотеку для генерації випадкових чисел.

# 1. АНАЛІЗ ГЕНЕРАЦІЇ ВИПАДНОВИХ ДАНИХ У СУЧАСНОМУ СВІТІ

## 1.1 Визначення та використання випадковості

У загальному вживанні випадковість - це явна або фактична відсутність шаблону або передбачуваності в подіях. Випадкова послідовність подій, символів або кроків часто не має порядку та не слідує зрозумілій моделі чи комбінації. Окремі випадкові події, за визначенням, непередбачувані, але якщо відомий розподіл ймовірностей, частота різних результатів повторюваних подій (або «випробувань») є передбачуваною. Наприклад, під час кидання двох кубиків результат будь-який окремий кидок є непередбачуваним, але сума 7, як правило, траплятиметься вдвічі частіше, ніж 4. З цієї точки зору, випадковість не є випадковістю; це міра невизначеності результату. Випадковість стосується понять випадковості, ймовірності та інформаційної ентропії[1].

У галузях математики, ймовірності та статистики використовуються формальні визначення випадковості. У статистиці випадкова змінна — це присвоєння числового значення кожному можливому результату простору подій. Ця асоціація полегшує ідентифікацію та обчислення ймовірностей подій. Випадкові величини можуть з’являтися у випадковій послідовності. Випадковий процес — це послідовність випадкових змінних, результати яких не слідують детерміністичному шаблону, а слідують еволюції, описаній розподілами ймовірностей. Ці та інші конструкції надзвичайно корисні в теорії ймовірностей і різних застосуваннях випадковості.

Випадковість найчастіше використовується в статистиці для позначення чітко визначених статистичних властивостей. Методи Монте-Карло, які покладаються на випадкові вхідні дані (наприклад, від генераторів випадкових чисел або генераторів псевдовипадкових чисел), є важливими техніками в науці, особливо в галузі обчислювальної техніки. За аналогією, методи квазі-Монте-Карло використовують генератори квазівипадкових чисел[2].

Випадковий відбір, якщо він тісно пов’язаний із простою випадковою вибіркою, — це метод відбору елементів (часто званих одиницями) із сукупності, де ймовірність вибору конкретного елемента є пропорцією цих елементів у сукупності. Наприклад, якщо миска містить лише 10 червоних кульок і 90 синіх кульок, механізм випадкового вибору вибере червону кульку з імовірністю 1/10. Зауважте, що механізм випадкового вибору, який вибрав 10 кульок із цієї чаші, не обов’язково призведе до 1 червоної та 9 синіх. У ситуаціях, коли генеральна сукупність складається з елементів, які можна розрізнити, механізм випадкового вибору вимагає рівних імовірностей для будь-якого елемента, який буде обрано. Тобто, якщо процес відбору такий, що кожен член сукупності, скажімо суб’єкти дослідження, має однакову ймовірність бути обраним, тоді можна сказати, що процес відбору є випадковим.

Відповідно до теорії Рамсі, чиста випадковість неможлива, особливо для великих структур. Математик Теодор Моцкін припустив, що «хоча безлад більш імовірний загалом, повний безлад неможливий». Нерозуміння цього може призвести до численних теорій змови. Крістіан С. Калуде заявив, що «враховуючи неможливість справжньої випадковості, зусилля спрямовані на вивчення ступенів випадковості». Можна довести, що існує нескінченна ієрархія (з точки зору якості або сили) форм випадковості.

Багато наукових галузей стурбовані випадковістю:

- Алгоритмічна ймовірність;
- Теорія хаосу;
- Криптографія;
- Теорія ігор;
- Теорія інформації;
- Розпізнавання образів;
- Теорія перколяції;
- Теорія ймовірностей;
- Квантова механіка;

- Випадкове блукання;
- Статистична механіка;
- Статистика.

**У фізичних науках.** У 19 столітті вчені використали ідею хаотичних рухів молекул у розвитку статистичної механіки для пояснення явищ термодинаміки та властивостей газів.

Відповідно до кількох стандартних інтерпретацій квантової механіки, мікроскопічні явища є об’єктивно випадковими. Тобто в експерименті, який контролює всі причинно значущі параметри, деякі аспекти результату все ще змінюються випадковим чином. Наприклад, якщо один нестабільний атом помістити в контрольоване середовище, неможливо передбачити, скільки часу знадобиться для розпаду атома — лише ймовірність розпаду за певний час. Таким чином, квантова механіка визначає не результат окремих експериментів, а лише ймовірності. Теорії прихованих змінних відкидають погляд на те, що природа містить незмінну випадковість: такі теорії стверджують, що в процесах, які здаються випадковими, властивості з певним статистичним розподілом діють за лаштунками, визначаючи результат у кожному випадку.

**В біології.** Сучасний еволюційний синтез приписує спостережуване розмаїття життя випадковим генетичним мутаціям, за якими слідує природний відбір. Останній зберігає деякі випадкові мутації в генофонді через систематично покращені шанси на виживання та розмноження, які ці мутовані гени надають особам, які ними володіють. Місцезнаходження мутації не зовсім випадкове, як, наприклад, біологічно важливі регіони можуть бути більш захищені від мутацій.

Кілька авторів також стверджують, що еволюція (а іноді й розвиток) потребує специфічної форми випадковості, а саме впровадження якісно нової поведінки. Замість вибору однієї можливості серед кількох заздалегідь заданих, ця випадковість відповідає утворенню нових можливостей.

Характеристики організму виникають певною мірою детерміновано (наприклад, під впливом генів і середовища) і певною мірою випадково.

Наприклад, густота веснянок, які з’являються на шкірі людини, контролюється генами та впливом світла; тоді як точне розташування окремих веснянок виглядає випадковим.

Що стосується поведінки, випадковість важлива, якщо тварина має поводитися непередбачувано для інших. Наприклад, комахи під час польоту мають тенденцію рухатися з випадковими змінами напрямку, що ускладнює хижакам, які переслідують їх, передбачити їх траєкторії.

**В математиці.** Математична теорія ймовірності виникла в результаті спроб сформулювати математичний опис випадкових подій, спочатку в контексті азартних ігор, але пізніше у зв'язку з фізикою. Статистичні дані використовуються для визначення основного розподілу ймовірностей набору емпіричних спостережень. Для цілей моделювання необхідно мати великий запас випадкових чисел або засоби для їх генерації на вимогу.

Алгоритмічна теорія інформації вивчає, серед інших тем, що являє собою випадкову послідовність. Основна ідея полягає в тому, що рядок бітів є випадковим тоді і тільки тоді, коли він коротший за будь-яку комп’ютерну програму, яка може створити цей рядок (випадковість за Колмогоровом), що означає, що випадкові рядки – це ті, які не можна стиснути. Піонерами цієї галузі є Андрій Колмогоров і його учень Пер Мартін-Леф, Рей Соломонов і Грегорі Чайтін. Що стосується поняття нескінченної послідовності, то математики зазвичай приймають напівепонімне визначення Пера Мартіна-Лефа: нескінченна послідовність є випадковою тоді і тільки тоді, коли вона протистоїть усім рекурсивно перелічуваним нульовим множинам. Інші поняття випадкових послідовностей включають, серед іншого, рекурсивну випадковість і випадковість Шнорра, які базуються на рекурсивно обчислюваних мартингалах. Йонге Ван показав, що ці поняття випадковості загалом різні.

Випадковість зустрічається в таких числах, як log(2) і pi. Десяткові цифри числа «пі» складають нескінченну послідовність і «ніколи не повторюються циклічно». Такі числа, як пі, також вважаються нормальними:

Пі, безсумнівно, поводиться таким чином. У перших шести мільярдах знаків після коми числа пі кожна цифра від 0 до 9 з’являється приблизно шістсот мільйонів разів. Однак такі результати, можливо, випадкові, не доводять нормальності навіть у 10-й основі, а тим більше нормальності в інших основах числення.

**У статистиці.** Числову послідовність називають статистично випадковою, якщо вона не містить розпізнаваних закономірностей або закономірностей; послідовності, такі як результати ідеального кидка кубика або цифри π, демонструють статистичну випадковість[3].

Статистична випадковість не обов’язково означає «справжню» випадковість, тобто об’єктивну непередбачуваність. Псевдовипадковість є достатньою для багатьох застосувань, таких як статистика, звідси й назва статистична випадковість.

Глобальна випадковість і локальна випадковість відрізняються. Більшість філософських концепцій випадковості є глобальними, оскільки вони базуються на ідеї, що «в довгостроковій перспективі» послідовність виглядає справді випадковою, навіть якщо певні підпослідовності не виглядатимуть випадковими. У «дійсно» випадковій послідовності чисел достатньої довжини, наприклад, ймовірно, що будуть довгі послідовності лише повторюваних чисел, хоча в цілому послідовність може бути випадковою. Локальна випадковість відноситься до ідеї, що можуть існувати мінімальні довжини послідовності, в яких випадкові розподіли апроксимуються. Довгі розрізи одних і тих самих чисел, навіть тих, що генеруються «справді» випадковими процесами, зменшать «локальну випадковість» вибірки (вона може бути локально випадковою лише для послідовностей з 10 000 чисел; взяття послідовностей менше ніж 1000 може не виглядати випадковим взагалі, наприклад)[4].

Таким чином, послідовність, що демонструє закономірність, не є статистично випадковою. Відповідно до принципів теорії Рамсі, досить великі об'єкти обов'язково повинні містити задану підструктуру («повний безлад неможливий»).

Законодавство щодо азартних ігор накладає на ігрові автомати певні стандарти статистичної випадковості.

**В інформатиці.** В інформатиці нерелевантні або безглузді дані вважаються шумом. Шум складається з численних перехідних збурень із статистично рандомізованим розподілом у часі.

У теорії зв'язку випадковість у сигналі називається «шумом» і протиставляється тому компоненту його зміни, який причинно пов'язаний з джерелом, сигналом.

З точки зору розвитку випадкових мереж, для комунікацій випадковість ґрунтується на двох простих припущеннях Пауля Ердеша та Альфреда Реньї, які сказали, що існує фіксована кількість вузлів, і ця кількість залишається фіксованою протягом усього життя мережі, і що всі вузли були рівними та зв'язані випадковим чином один з одним.

**У фінансах.** Гіпотеза випадкового блукання передбачає, що ціни активів на організованому ринку змінюються випадковим чином, у тому сенсі, що очікувана вартість їх зміни дорівнює нулю, але фактична вартість може виявитися позитивною або негативною. Загалом, на ціни активів впливають різні непередбачувані події в загальному економічному середовищі.

**В політиці.** У деяких юрисдикціях випадковий відбір може бути офіційним методом розв'язання результатів голосування з рівними результатами. Його використання в політиці бере свій початок дуже давно. Багато посад у Стародавніх Афінах обиралися шляхом жеребкування замість сучасного голосування.

### 1.1.1 Тестування випадковсті

Тест на випадковість (або тест на випадковість) в оцінці даних — це тест, який використовується для аналізу розподілу набору даних, щоб побачити, чи можна його описати як випадковий (без шаблону). У стохастичному моделюванні, як і в деяких комп'ютерних симуляціях, очікувану випадковість потенційних вхідних даних можна перевірити за допомогою формального тесту на випадковість, щоб показати, що дані дійсні для використання в моделюванні. У деяких випадках дані виявляють очевидну невипадкову закономірність, як-от у випадку так званих

«запусків даних» (наприклад, очікування випадкових 0–9, але знаходження «4 3 2 1 0 4 3 2 1...» і рідко вище 4). Якщо вибраний набір даних не пройшов тести, параметри можна змінити або використати інші рандомізовані дані, які проходять тести на випадковість.

На практиці використовується досить невелика кількість різних типів генераторів (псевдо)випадкових чисел.

- Узагальнений генератор Фібоначчі;
- Криптографічні генератори;
- Квадратичний конгруентний генератор;
- Генератори клітинних автоматів;
- Псевдовипадкова двійкова послідовність.

Ці різні генератори мають різний ступінь успіху в проходженні прийнятих наборів тестів. Кілька широко використовуваних генераторів провалили випробування більш-менш погано, тоді як інші «кращі» та попередні генератори (у тому сенсі, що вони пройшли всі поточні випробування та вже існували) були здебільшого проігноровані.

Існує багато практичних мір випадковості для двійкової послідовності. До них належать показники, засновані на статистичних перевірках, перетвореннях і складності або їх суміші. Добре відомим і широко використовуваним збірником тестів був Diehard Battery of Tests, представлений Marsaglia; це було розширено до набору TestU01 L'Ecuyer і Simard. Використання перетворення Адамара для вимірювання випадковості було запропоновано С. Каком і розвинене далі Філліпсом, Юеном, Хопкінсом, Бет і Даєм, Мундом, Марсалією і Заманом.

Деякі з цих тестів, які мають лінійну складність, забезпечують спектральні показники випадковості.

## 1.2 Історія появи та застосування випадкових чисел у комп’ютерній техніці

В природі дуже легко отримати випадкове число зазвичай вони є наслідками дуже складних фізичних процесів які дуже важко змоделювати на комп’ютері. Одними із перших генераторів випадкових чисел які придумало людство це - гральні кубики. Для отримання випадкового числа їх потрібно підкинути після чого кубик упаде на стіл якоюсь своєю стороною і тим самим в каже яке саме випадкове число було отримано. Гральні кубики(1.1) є найпростішим одним із самих давніх способів отримання випадкових чисел[5].

Рисунок 1.1 - Гральні кубики

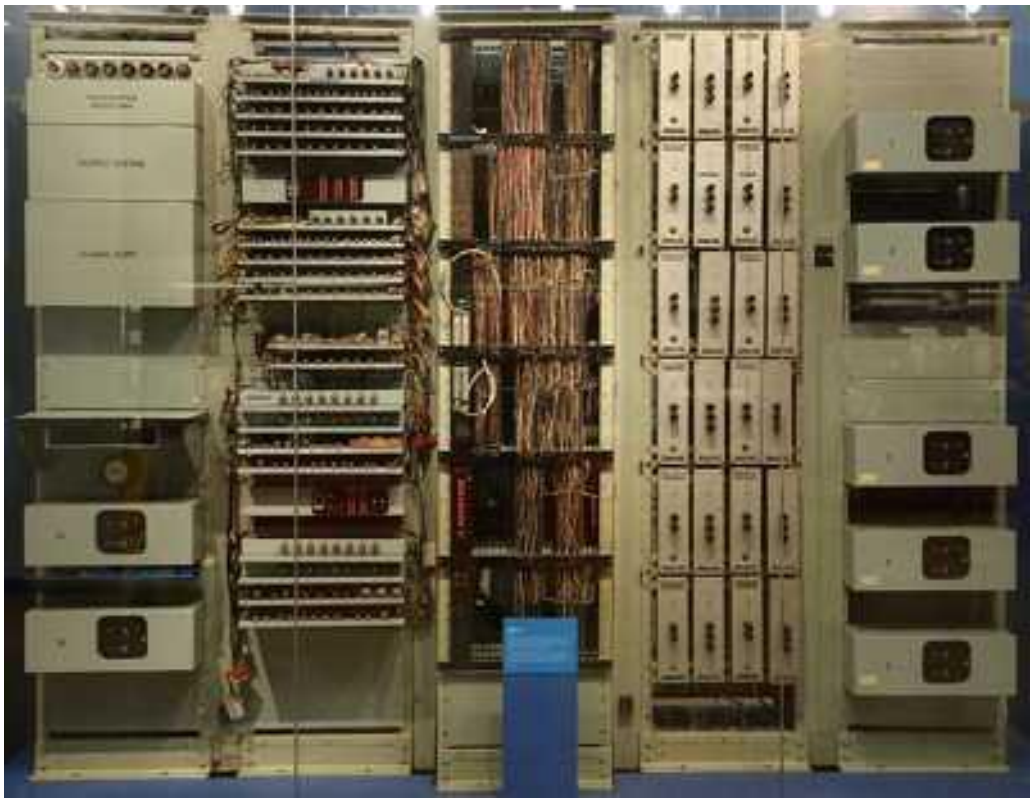
На відміну від природи у комп’ютерній техніці дуже важко отримати випадкові числа тому що комп’ютерні системи зазвичай є детермінованими алгоритмічними системами. Яким не можна сказати згенеруй мені випадкове число. Тому на початку розвитку комп’ютерної техніки проблема генерації випадкових буда доволі серйозною.

### 1.2.1 Перша машина генерації випадкових чисел

У сорокових роках двадцятого століття американський аналітичний центр RAND створив першу електрону машину яка могла генерувати випадкові числа. Принцип її роботи був схожий на електрону рулетку яка підключена до компютеру. Перед використанням машину було ретельно відфільтровані та протестовані після чого було створено таблицю з випадковими числами. У подальшому послідовність чисел яку згенерувала ця машина будо збережено та занесено в книгу яка називається “Один міліон випадкових чисел сто тисяч нормальних відхилень”. На сьогоднішній день цю книгу можна придбати вона була перевидана у 2001 році.

### 1.2.2 ERNIE.

Також десь в той самий час у Британії була побудована схожа машина. ERNIE - це апаратний генератор випадкових чисел. Вперше він був створений у британііі на поштово дослідній станції командою під керівництвом Сидни Бродхерста(1.2).



Малюнок 1.2 – ERNIE[6].

Ця машина використовувалась для генерації випадкових чисел для британської лотереї. Генерація чисел відбувалась за допомогою сигнального шуму, створюваного неоновими трубками. ERNIE генерував 2000 номерів за годину і був розміром з фургоном. У подальшому ця машина ставала більш компактною та працювала більш швидко. В результаті розвідку цієї машини тепловий шум від ламп було замінено на тепловий шум в транзисторах в якості джерела випадковості для генерації істинних випадкових чисел. У сучасній версії цієї машини використовують квантовий генератор випадкових чисел.

### 1.2.3 Перший персональний комп’ютер с генераторам випадковості.

Ferranti Mark 1 – був створений у 1951 році та мав вбудований апаратний генератор випадкових чисел. Генератор вбудований в цей компютер генерував послідовність із двадцяти випадкових бітів на основі дробового шуму. Цей метод був розроблений Аланом Тюрінгом.  Але цей генератор був далеким від ідеалу. Комп’ютерна техніка тих часів і так працювала не стабільно а цей генератор вносив

ще більше нестабільносте в систему на ньому неможливо було добитися відтворюваності результатів через що написання коду з використанням цього генератору було дуже складним.

### 1.2.4 Поява та розвиток генераторів псевдо випадкових чисел

Дуже швидко стало зрозуміло що для комп'ютерної технікі потрібні алгоритми генерації послідовності чисел які будуть мати властивості випадкової послідовності але при однакових параметрах алгоритму можна буде отримати однаковий результат. Першим хто розробив генератор псевдовипадкових чисел був Джон фон Нейман. Алгоритм якій він запропонував називається метод середини квадратів. Він полягає у тому що потрібно почати з деякого числа, взяти його квадрат, відкинути цифри з середини результату, знову взяти квадрат і відкинути середину і т.д. Таким чином ми отримуємо послідовність чисел яка схожа на випадкову але ми можемо її відтворити використавши таке саме початкове число. Хоч цей алгоритм і має велике історичне значення він має дуже великий недолік алгоритм дуже швидко зациклюється.

Оскілки повністю запобігти появі циклів не можливо при використані детермінованих функцій які використовують в якості вхідних параметрів дані які були отримані при попередній ітерації алгоритми генерації почали шукати шляхи збільшення періоду до зациклення алгоритму. Лінійний конгруентний метод один із алгоритмів період зациклення якого дуже довгий.

Наступним витком у розвитку ПВГЧ стала поява протоколу SSL для якого потрібні були випадкові числа. На початку розробники генераторів не переймалися питаннями безпеки через що з'явилася велика купа дір в безпеці. Через це з'явилося поняття криптографічно стійкого ГПВЧ. Криптографічно стійкий ГПВЧ повинен відповідати великому списку вимог які зводяться до того що маючи велику вибірку чисел зцентрованих алгоритмом вгадати наступний елемент повинно бути важко або взагалі не можливо.

### 1.2.5 Розвиток апаратних генераторів випадкових чисел

Паралельно з ПВГЧ розвивались і апаратні генератори випадкових чисел. Це пристрої які генерують випадкові числа спираючись на надійні джерела ентропії. Такі генератори зазвичай вимірюють фізичні явища що не можливо або дуже складно прорахувати на комп'ютері. Апаратні алгоритми розвивалися в бік збільшення швидкості генерації та зменшення габаритів самого модулю. Най сучаснішим апаратним генератором випадковості можна вважати квантові генратори.

## 1.3 Апаратні генератори випадкових чисел

Апаратний генератор випадкових чисел (генератор істинно випадкових чисел) - пристрій, який генерує послідовність випадкових чисел на основі вимірюваних параметрів фізичного процесу, що протікає. Робота таких пристроїв часто заснована на використанні надійних джерел ентропії, таких як тепловий шум, фотоефект, квантові явища тощо. Ці процеси теоретично абсолютно непередбачувані, на практиці ж одержувані з них випадкові числа можна перевірити за допомогою спеціальних статистичних тестів[7].

Апаратний генератор випадкових коливань зазвичай складається з перетворювача для перетворення деяких аспектів фізичного явлення в електричний сигнал, підсилювача та інших електронних схем для збільшення амплітуди випадкових коливань до вимірюваного рівня, а також іншого типу аналого-цифрового перетворювача. цифровий перетворювач для перетворення вихідних даних у цифрове число, часто в простій подвійній цифрі 0 або 1. Методом багатократного вибору випадково змінюваного сигналу отримується серія випадкових чисел.

### 1.3.1 Фізичні явища що використовують для генерації випадкових чисел

Фізичні явища що використовуються при генерації можуть мати квантовий характер або фізичний характер який можна назвати класичним тобто явища що не мають квантових ознак.

При квантовій генерації випадкових чисел існує два фундаментальних джерела практичної квантово-механічної фізичної випадковості: квантова механіка на атомному або субатомному рівні та тепловий шум. Квантова механіка передбачає, що певні фізичні явища, такі як ядерний розпад атомів, є фундаментально випадковими і, в принципі, не можуть бути передбачені, оскільки світ існує при температурі вище абсолютного нуля, кожна система має деякі випадкові варіації свого стану; наприклад, молекули газів, що утворюють повітря, постійно відскакують одна від одної випадковим чином. Ця випадковість також є квантовим явищем[8].

Оскільки результати квантово-механічних подій неможливо передбачити навіть у принципі, вони є «золотим стандартом» для генерації випадкових чисел. Деякі квантові явища, які використовуються для генерації випадкових чисел, включають:

**Дробовий шум**, джерело квантово-механічного шуму в електронних схемах. Простий приклад - лампа, що світить на фотодіоді. Через принцип невизначеності фотони, що надходять, створюють шум у ланцюзі. Збір шуму для використання викликає деякі проблеми, але це особливо просте випадкове джерело шуму. Однак енергія дробового шуму не завжди добре розподіляється по всій смузі пропускання. Електронні трубки газового діода та тиратрона в поперечному магнітному полі можуть генерувати значну шумову енергію (10 вольт або більше у навантаженні з високим імпедансом), але мають дуже піковий розподіл енергії та вимагають ретельної фільтрації для досягнення рівності в широкому спектрі.

**Джерело випромінювання ядерного розпаду**, виявлене лічильником Гейгера (1.3), підключеним до ПК.


Малюнок 1.3 – лічильник Гейгера[9]

**Фотони що проходять через напівпрозоре дзеркало**. Взаємовиключні події (відображення/передача) виявляються та пов’язуються з бітовими значеннями «0» або «1» відповідно[10].

**Підсилення сигналу здійснюється на базі транзистора зі зворотним зміщенням**. Емітер насичений електронами, і іноді вони тунелюють через заборонену зону та виходять через базу. Потім цей сигнал посилюється ще кількома транзисторами, а результат подається на тригер Шмітта.

**Спонтанне параметричне пониження**, що призводить до вибору бінарного фазового стану в виродженому оптичному параметричному осциляторі.

**Флуктуації енергії вакууму**, виміряні за допомогою гомодинного виявлення.

Класичні фізичні явища легше виявити. Вони певною мірою вразливі до атак через зниження температури системи, хоча більшість систем припиняють працювати при температурах, достатньо низьких, щоб зменшити шум у два рази Деякі з використовуваних теплових явищ включають:

**Тепловий шум від резистора**,(1.4) посилений для забезпечення джерела випадкової напруги.

Малюнок 1.4 – Резистор.

**Лавинний шум**, створюваний лавинним діодом(1.5), або шум пробою Зенера від стабілітрона зі зворотним зміщенням.

Малюнок 1.5 – Лавинний діод

**Атмосферний шум**, виявлений радіоприймачем, підключеним до комп’ютера.

За відсутності квантових ефектів або теплового шуму можна використовувати інші явища, які мають тенденцію бути випадковими, хоча й такими способами, які важко охарактеризувати законами фізики. Коли декілька таких джерел ретельно об’єднуються таким чином можна зібрати достатню кількість ентропії для створення криптографічних ключів і одноразових кодів, хоча, як правило, з обмеженою швидкістю. Перевага полягає в тому, що цей підхід, в принципі, не потребує спеціального обладнання. Недоліком є те, що достатньо обізнаний зловмисник може таємно змінити програмне забезпечення або його вхідні дані, таким чином суттєво зменшуючи випадковість результату. Основним джерелом випадковості, який зазвичай використовується в таких підходах, є

точний час переривань, викликаних механічними пристроями введення/виведення, такими як клавіатури та дисководи, різні лічильники системної інформації тощо.

Іншим змінним фізичним явищем, яке легко виміряти, є дрейф годинника. Це явище полягає у тому що навіть най точніше годинники не будуть працювати синхронно. Існує кілька способів вимірювання та використання дрейфу годинника як джерела випадковості.

Одним із способів створення апаратного генератора випадкових чисел є використання двох незалежних тактових генераторів, один з яких, наприклад, цокає 100 разів на секунду, а інший - 1 мільйон разів на секунду. У середньому швидший кристал цокатиме 10 000 разів за кожен тик повільнішого. Але оскільки кристали годинника не точні, точна кількість тактів варіюватиметься. Цей варіант можна використовувати для створення довільних бітів. Наприклад, якщо кількість швидких тиків парне, вибирається 0, а якщо кількість тиків непарне, вибирається Таким чином, така схема 100/1000000 ГСЧ може виробляти 100 кілька випадкових бітів за секунду. Зазвичай така система зміщена - вона може, наприклад, виробляти більше нулів, ніж одиниць - і тому сотні кілька випадкових бітів "знебарвлюються" для отримання декількох незміщених бітів.

Існує також аналогічний спосіб створення свого роду програмного генератора випадкових чисел. Це включає порівняння тика таймера операційної системи (такт, який зазвичай становить 100-1000 разів в секунду) і швидкості процесора . Якщо таймер ОС і ЦП працюють на двох незалежних тактових генераторах, ситуація ідеальна і більш менш аналогічна попередньому прикладу. Але навіть якщо вони обидва використовують один і той же тактовий кристал, процес/програма, що виконує вимірювання дрейфу годинника, «обурюється» багатьма більш-менш непередбачуваними подіями в ЦП, такими як переривання та інші процеси та програми, які виконуються одночасно. Таким чином, вимірювання, як і раніше, даватиме досить добрі випадкові числа.

Більшість апаратних генераторів випадкових чисел, як описані вище, досить повільні. Тому більшість програм використовують їх тільки для створення хорошого початкового числа, яке потім передається генератору псевдовипадкових

чисел або криптографічно безпечного генератора псевдовипадкових чисел для швидкого створення безлічі випадкових чисел.

### 1.3.2 Використання видимих подій

Розробники програмного забезпечення, які не мають справжніх генераторів випадкових чисел, часто намагаються розробити їх, вимірюючи фізичні події, доступні програмному забезпеченню. Прикладом є вимірювання часу між натисканнями клавіш користувачем, а потім вибір молодшого біта лічильника як випадкової цифри. Аналогічний підхід вимірює планування завдань, мережеве звернення, час пошуку головки диска та інші внутрішні події. Один проект Microsoft включає дуже довгий список таких внутрішніх значень, форму криптографічно безпечного генератора псевдовипадкових чисел.

Цей метод є ризикованим, коли він використовує події, керовані комп'ютером, тому що розумний зловмисник може передбачити криптографічний ключ, контролюючи зовнішні події. Це також ризиковано, оскільки подія, генерована користувачем (наприклад, натискання клавіші), може бути підроблена досить винахідливим зловмисником, що дозволить контролювати «випадкові значення», використовувані криптографією.

Однак при належній обережності можна розробити систему, яка виробляє криптографічно безпечні випадкові числа джерел випадковості, доступних в сучасному комп'ютері. Базова схема полягає у підтримці «ентропійного пулу» випадкових бітів, які, як передбачається, невідомі зловмиснику. Нова випадковість додається щоразу, коли це можливо (наприклад, коли користувач натискає клавішу), і зберігається оцінка кількості бітів у пулі, яка не може бути відома зловмиснику.

### 1.3.3 Проблеми та недоліки апаратних генераторів

Дуже легко неправильно сконструювати апаратні чи програмні пристрої, які намагаються генерувати випадкові числа. Крім того, більшість з них «ламаються» непомітно, часто роблячи все більш випадкові числа в міру їхньої деградації.

Фізичним прикладом може бути радіоактивність димових сповіщувачів, що швидко зменшується, згаданих раніше, якби це джерело використовувалося безпосередньо. Режими відмови в таких пристроях багаточисельні, вони складні, повільні та важко виявити. Методи, що поєднують кілька джерел ентропії, надійніші.

Оскільки багато джерел ентропії часто досить тендітні і безшумно виходять з ладу, статистичні перевірки їх вихідних даних повинні виконуватися постійно. Багато, але не всі такі пристрої включають деякі такі тести в програмне забезпечення, яке зчитує пристрій.

### 1.4 Програмні генератори випадкових чисел

Генератор псевдовипадкових чисел, також відомий як детермінований генератор випадкових бітів, являє собою алгоритм генерації послідовності чисел, властивості якої наближаються до властивостей послідовностей випадкових чисел. Генерована ГПВЧ послідовність не є дійсно випадковою , тому що вона повністю визначається початковим значенням, що називається початковим числом ГПВЧ (яке може включати дійсно випадкові значення). Хоча послідовності, ближчі до істинно випадкових, можна генерувати за допомогою апаратних генераторів випадкових чисел, генераторів псевдовипадкових чисел важливі на практиці через їх швидкість генерації чисел та їх відтворюваності.

Хороші статистичні властивості є головною вимогою до ГПВЧ. Зазвичай потрібен математичний аналіз щоб впевнитись що ГПВЧ генерує числа, які є досить близькими до випадкових.

Існує кілька основних методів генерації псевдо випадкових чисел всі генератори або використовують один із них або використовують різні комбінації із них. До цих методів відносяться: лінійний конгруентний метод, Алгоритм Mother-of-All, вихор Мерсенна, Алгоритм Xorshift.

### 1.4.1Основні методи генерації випадкових чисел

**Лінійний конгруентний генератор** (ЛКГ) — це алгоритм, який дає послідовність псевдорандомізованих чисел, обчислених за допомогою розривного кусково-лінійного рівняння. Метод являє собою один із найстаріших і найвідоміших алгоритмів генератора псевдовипадкових чисел. Теорію, що лежить в їх основі, відносно легко зрозуміти, і вони легко і швидко реалізуються, особливо на комп'ютерному обладнанні, яке може забезпечити модульну арифметику шляхом скорочення бітів пам'яті[11].

Генератор Лемера був опублікований у 1951 році, а лінійний конгруентний генератор був опублікований у 1958 році В. Е. Томсоном і А. Ротенбергом. Перевага ЛКГ полягає в тому, що відповідний вибір параметрів призводить до періоду, який є відомим і довгим. Хоча це не єдиний критерій але занадто короткий період є фатальним недоліком ранніх генераторів псевдовипадкових чисел. Перевага ЛКГ полягає в тому, що правильно підібрані параметри гарантують довгий період генерації.

У той час як ЛКГ здатні створювати псевдовипадкові числа, які можуть пройти формальні тести на випадковість, результат надзвичайно чутливий до вибору параметрів m і a. Наприклад, a = 1 і c = 1 створює простий лічильник за модулем m, який має довгий період, але, очевидно, невипадковий.

Навіть маючі всі ці недоліки ЛКГ використовують до сих пір тому що він легкий у реалізації відносно швидкий і не потребує багато пам'яті для генерації псевдовипадкових чисел.

**Регістр зсуву з лінійним зворотним зв'язком** (Linear-feedback shift register або LFSR) це – це регістр зсуву, вхідний біт якого є лінійною функцією його попереднього стану[11].

Найпоширенішою лінійною функцією окремих бітів є виключаюче або (XOR). Таким чином, LFSR найчастіше є регістром зсуву, вхідний біт якого керується XOR деяких бітів загального значення регістра зсуву.

Початкове значення LFSR називається початковим, і оскільки робота регістра є детермінованою, потік значень, створених регістром, повністю

визначається його поточним (або попереднім) станом. Подібним чином, оскільки регістр має кінцеву кількість можливих станів, він повинен зрештою увійти в повторюваний цикл. Однак LFSR із добре підібраною функцією зворотного зв'язку може створити послідовність бітів, яка виглядає випадковою та має дуже довгий цикл.

Застосування LFSR включає генерування псевдовипадкових чисел, псевдо шумових послідовностей, швидких цифрових лічильників і послідовностей відбілювання. Як апаратна, так і програмна реалізації LFSR є поширеними.

Математика перевірки циклічної надлишковості, яка використовується для забезпечення швидкої перевірки на наявність помилок передачі, тісно пов'язана з математикою LFSR. Можна створити відносно складну логіку за допомогою простих будівельних блоків.

**Вихор Мерсенна** — це генератор псевдовипадкових чисел загального призначення (PRNG), розроблений у 1997 році Макото Мацумото і Такудзі Нісімура Його назва походить від того факту, що довжиною періоду вибрано просте число Мерсенна[12].

Вихор Мерсенна був розроблений спеціально для усунення більшості недоліків, виявлених у старих ПВГЧ. Найпоширеніша версія алгоритму Мерсенна Твістер заснована на простому числі Мерсенна. Стандартна реалізація, використовує 32-бітну довжину слова. Існує ще одна реалізація, яка використовує 64-бітну довжину слова він генерує іншу послідовність.

Є декілька модифікацій цього алгоритму:

- CryptMT — це потоковий шифр і криптографічно захищений генератор псевдовипадкових чисел, який внутрішньо використовує Mersenne Twister.
- MTGP — це варіант Mersenne Twister, оптимізований для графічних процесорів, опублікований Муцуо Сайто та Макото Мацумото.
- TinyMT — це варіант Mersenne Twister, запропонований Сайто та Мацумото в 2011 році. TinyMT використовує лише 127 біт простору

стану, що є суттєвим зменшенням у порівнянні з 2,5 КіБ стану в оригіналі.

Перевагами даного алгоритму є те що він проходить всі того часині та деякі сучасні перевірки на статистичну випадковість. Ще одною важливою перевагою є велика довжина періоду генеруємих чисел. Цей алгоритм є дуже швидким і його реалізація зазичай є швидшою за деякі апаратні генератори випадкових чисел.

Хоча алгоритм і використовують на сьогоднішній день він має купу недоліків таких як велика кількість пам’яті яка потрібна для роботи алгоритму. По сьогоднішнім міркам цей алгоритм має малу пропускну здатність. Це одним недоліком є те що алгоритм має декілька вразливостей та не може використовуватися для криптографії.

**Генератори випадкових чисел Xorshift**, так звані генераторами регістрів зсуву, є класом генераторів псевдовипадкових чисел, які були винайдені Джорджем Марсалією. Вони являють собою підмножину регістрів зсуву з лінійним зворотним зв'язком (LFSR), які дозволяють особливо ефективну реалізацію в програмному забезпеченні без надмірного використання розріджених поліномів. Вони генерують наступне число у своїй послідовності, неодноразово беручи ексклюзивне число або його версію з бітовим зсувом. Це робить виконання надзвичайно ефективним на сучасних комп’ютерних архітектурах, але це не впливає на ефективність апаратної реалізації. Як і всі LFSR, параметри потрібно вибирати дуже ретельно, щоб досягти тривалого періоду[13].

Для виконання в програмному забезпеченні генератори xorshift є одними з найшвидших не криптографічно захищених генераторів випадкових чисел, які вимагають дуже маленького коду та стану. Однак вони не проходять усі статистичні перевірки без подальшого уточнення. Ця слабкість виправляється шляхом поєднання їх із нелінійною функцією, як описано в оригінальній статті. Оскільки звичайні генератори xorshift (без нелінійного кроку) не проходять деякі статистичні тести, їх звинувачують у ненадійності.

## 1.5 Використання генераторів випадкових чисел та вимоги до них

На сьогоднішній день галузей застосування ГПВЧ багато але одним із головних напрямків де використовуються генератори це криптографія.

### 1.5.1 Використання генераторів випадкових чисел у галузі криптографії

В галузі криптографії генератори випадкових чисел застосовують безпосередньо для генерації ключів. Генерація ключів це найнижчий рівень захисту якщо ключ буде скомпрометовано усі інші рівні захисту будуть повністю марні. Через що до генераторів які використовуються у цій галузі висувають жорсткі вимоги. З розвитком генераторів випадкових чисел було сформовано поняття Криптографічно безпечного генератору випадкових чисел. Для того щоб генератор вважався КБГВЧ він повинен відповідати критеріям які були сформовані за час розвитку генераторів випадкових чисел.

Вимоги КБГВЧ діляться на дві групи: по-перше, вони повинні пройти статистичні тести на випадковість; а по-друге, що вони повинні добре витримувати серйозну атаку, навіть коли частина їхнього початкового чи робочого стану стає доступною для зловмисника.

Кожен КБГВЧ повинен задовольняти наступний бітовий тест. Тобто, враховуючи перші k бітів випадкової послідовності, не існує алгоритму з поліноміальним часом, який міг би передбачити (k+1)-й біт з імовірністю успіху, що не є незначно кращою, ніж 50%. Ендрю Яо довів у 1982 році, що генератор, який пройшов тест наступного біта, пройде всі інші поліноміальні статистичні тести на випадковість.

Кожний КБГВЧ повинен витримувати "компромісне розширення стану". У випадку, якщо частково або повністю його стан було розкрито (або вгадано правильно), неможливо буде реконструювати потік випадкових чисел до відкриття. Крім того, якщо під час роботи є вхідні дані ентропії, використання знань про стан вхідних даних для прогнозування майбутніх умов стану КБГВЧ має бути неможливим.

Приклад: якщо КБГВЧ, який розглядається, створює вихідні дані шляхом обчислення бітів π у послідовності, починаючи з деякої невідомої точки у двійковому розширенні, він цілком може задовольнити перевірку наступного біта і, отже, бути статистично випадковим, оскільки π виглядає як випадкова послідовність . (Це буде гарантовано, якщо π є звичайним числом, наприклад.) Однак цей алгоритм не є криптографічно безпечним; зловмисник, який визначає, який біт числа pi (тобто стан алгоритму) зараз використовується, зможе також обчислити всі попередні біти.

Більшість ГПВЧ не підходять для використання як КБГВЧ і не вгадуються з обох причин. По-перше, хоча більшість виходів ГПВЧ здаються випадковими для різних статистичних тестів, вони не протистоять визначеній зворотній інженерії. Можуть бути знайдені спеціальні статистичні тести, спеціально налаштовані на такий PRNG, який показує, що випадкові числа не є справді випадковими. По-друге, для більшості PRNG, коли їхній стан було виявлено, усі минулі випадкові числа можна відтворити, дозволяючи зловмиснику прочитати всі минулі повідомлення, а також майбутні.

На генератори випадкових чисел існують типові атаки при проектуванні генераторів потрібно враховувати. Атаки на програмні генератори та на апаратні генератори відрізняються.

Атаки на апаратні генератори зазвичай це атаки на сам пристрій зловмисники зазвичай хочуть отримати доступ до джерела випадковості яке використовує пристрій. Наприклад отримати інформацію про переривання жорсткого диску чи отримання доступу до радіо каналу який використовуються при генерації.

Існує декілька основних атак на програмні генератори. Пряма крипто аналітична атака коли зловмисник отримав частину потоку випадкових бітів і може використовувати це, щоб відрізнити вихід RNG від справді випадкового потоку. Атаки на основі вхідних даних змінити вхідні дані RNG, щоб атакувати їх, наприклад, «видимивши» наявну ентропію з системи та перевівши її у відомий стан. Атаки розширення компрометації стану коли внутрішній секретний стан RNG відомий у певний час, використовуйте це для прогнозування майбутніх виходів або

відновлення попередніх виходів. Це може статися, коли генератор запускається і має невелику ентропію або взагалі її не має (особливо якщо комп’ютер щойно завантажено та виконує дуже стандартну послідовність операцій), тому зловмисник може отримати початкове припущення про стан.

### 1.5.2 Генератори випадкових чисел у імітаційному моделюванні

Імітаційне моделювання – це процес створення та аналізу цифрового прототипу фізичної моделі для прогнозування її продуктивності в реальному світі. Імітаційне моделювання використовується, щоб допомогти дизайнерам та інженерам зрозуміти, чи може деталь вийти з ладу, за яких умов і яким чином, і які навантаження вона може витримати. Імітаційне моделювання також може допомогти передбачити потік рідини та моделі теплопередачі. Він аналізує приблизні робочі умови за допомогою програмного забезпечення для моделювання.

Генератори випадкових чисел використовують для перевірки як модель буде реагувати на різні варіанти подій та збору статистичних даних. До генераторів які використовуються для імітаційного моделювання висувають прості вимоги. Головною з них є можливість відтворити випадкову послідовність якщо це буде потрібно. Через це частіше всього в цій галузі використовують програмні генератори випадкових чисел через те що більшість з них детерміновані. Другою важливою вимогою є довжина періоду. Оскільки для дослідження властивостей може знадобитися велика кількість даних період генерації у алгоритму повинен бути дуже великим. Ще одним важливим параметром для імітаційного моделювання є наближеність генерованої послідовності до дійсно випадкових послідовностей чисел.

### 1.5.3 Генератори випадкових чисел у розробці ігор

Розробка ігор це галузь яка включає в себе велику кількість різних видів діяльності. Сюди відносяться як чисто технічні спеціальності такі як написання коду так і спеціальності більш творчі наприклад написання сценаріїв чи музики.

Генератори випадкових чисел можуть використовуватися одразу для кількох не пов’язаних завдань і вимоги до них відрізняються. В галузі розробки ігор це використання для генерації ігрових світів так і використання генераторів випадкових чисел у коді самої гри для рандомізаціїї ігрових елементів.

Створення ігрового рівня це складний процес який потребує багато часу. Якщо рівень дуже великий може знадобитися багато людей і часу для його створення і деталізації. Тому в ігровій індустрії існують інструменти які спрощують роботу дизайнерів за рахунок того що генерують рівень використовуючи спеціальні алгоритми. Деякі ігри повністю генерують свої ігрові рівні запомогою алгоритмів це ігри з так процедурною генерацією. Ігрові рівні в таких іграх унікальні для кожного гравця. Окрім ігор з процедурною генерацією рівні що генерує алгоритм можуть використовувати також для зменшення кількості роботи дизайнерів наприклад якщо потрібно згенерувати багато рівнів або один дуже великий. В такому випадку алгоритм генерує рівень а після цього дизайнер його допрацьовує тим самим кількість роботи зменшується.

Генерація рівнів зазвичай потребує повністю свого алгоритму який вбудований в сам генератор. Такі алгоритми генерації зазвичай мають параметри за допомогою яких можна вказати якість основні риси генеруємого рівня. Ці генератори також повинні мати можливість відтворити рівень маючи всі початкові параметри тобто він повинен бути детермінованим.

Також в галузі компютерних ігор генератори використовують в самому коді ігрового двигуна чи окремих модулів. За рахунок випадкових чисел  розробники додають елементи випадковості у гру. Це може бути будь що від розташуванням предметів на рівні до поведінки персонажів в середні гри. Існують ігри в яких майже все генерується за допомогою генераторів припадкових чисел. Також генератори випадкових чисел використовують для симуляції класичних ігор які існували до появи комп’ютерів і використовували гральні кубики чи карти.

Якщо генератори використовуються в середині гри однією головних вимог до ПВГЧ є цього компактність та навантаження на систему. Сучасні ігри зазвичай і так діже сильно навантажують систему і розробники не можуть собі дозволити

складні алгоритми генерації. Також деякі ігри потребують дуже швидкого генерування чисел.

### 1.5.4 ГПВЧ в галузі вимірювальної техніки

Галузь вимірювальної техніки це галузь яка займається вироблення там розробкою високоточного вимірювального обладнання. Це обладнання має проводити заміри фізичних явищ з високою точністю. Генератори випадкових чисел застосовують для тестів вироблених схем. Через те що схема повинна робити заміри з великою точною неможливо провести тести абсолютно всіх значень які може приймати схема. Через це за допомогою ГПВЧ генерують кінцевий набір значень та подають на плату після чого дивяться на реакцію якщо плата відреагувала нормально то її вважають готовою до роботи.

Для генераторів в цій галузі вадливо мати можливість відтворити результат щоб з’ясувати що саме в схемі вийшло з ладу. Іншим критерієм є швидкість генерації чисел так як за день може бути потрібно протестувати велику кількість плат.

# 2 ВИБІР АЛГОРИТМУ ТА ЙОГО МОДИФІКАЦІЯ

## 2.1 Вибір алгоритму генерації випадкових чисел

### 2.1.1 Вибір критеріїв для порівняльного аналізу.

Для того щоб з’ясувати який із алгоритмів підходить для модифікації потрібно порівняти їх за критеріями які необхідні для генерації чисел з нестандартних даних.

**Залежність якості алгоритму від параметрів.** Так як параметри алгоритму будуть генеруватися з отриманих даних нам важливо щоб алгоритм працював з якомога більшою різновидністю параметрів.

**Кількість початкових даних необхідна для генерації**. Най важливішим для виконання цієї роботи є кількість вхідних параметрів алгоритму. Тобто потрібно з’ясувати скільки пам’яті займає ключ кожного із алгоритмів а також кількості параметрів які використовуються для генерації.

**Довжина генеруємого періоду**. Для кожного генератора важим параметром є довженна генеруємого періоду. Хоча довжина періоду не є показником гарного алгоритму але якщо період генерації чисел замалий це є фатальним недоліком для алгоритму.

**Криптографічна стійкість**. Так як нам потрібно щоб модифікований алгоритм можна будо застосовувати в максимально можливій кількості галузей то потрібно відразу з’ясувати чи алгоритм який буде модифіковано відповідає вимогам криптографічно стійких алгоритмів.

**Програмна реалізація алгоритму.** Важливо з’ясувати наскільки складною э програмна реалізація ПВГЧ.

**Навантаження на систему.** Кількість ресурсів що необхідні для роботи алгоритму.

Іншими параметрами які можуть нас зацікавити: **Швидкість генерації чисел, статистична якість**, **важкість передбачення**, **відтворення результатів**.

В таблиці 2.1 представлені всі критерії та ступінь їх важливості.

Таблиця 2.1 - Критеріїв порівння алгоритмів та їх важливість

| Назва критерію | Важливість | Бажані значення |
|---|---|---|
| Залежність якості алгоритму від параметрів | Дуже вадливий | Низька залежність якості від параметрів |
| Кількість початкових даних необхідна для генерації | Має значення | Чим менше параметрів тим краще |
| Довжина генеруємого періоду | Не критична | Головне щоб період не був замалий |
| Криптографічна стійкість | Не критична | Добре якщо алгоритм криптографічно стійкий |
| Програмна реалізація алгоритму | Має значення | Чим простіше алгоритм в реалізації тим краще |
| Навантаження на систему | Має значення | Чим менше тим краще |
| Швидкість генерації | Не критична | Чим швидше тим краще |
| Статистична якість | Не критична | Максимальна схожість на Випадковий розподіл |
| Важкість передбачення | Не критична | Чим більше тим краще |
| Відтворення результатів | Не критична | Так |

### 2.1.2 Аналіз Алгоритмів для порівння

Для початку потрібно обрати алгоритм за допомогою якого буде виконуватися генерація чисел на основі нестандартних даних. Для вибору алгоритму проведемо порівняльний аналіз сучасних алгоритмів ГПВЧ. Для аналізу були обрані най сучасніші модифікації відомих алгоритмів які використовуються для генерації випадкових чисел а саме:

- Xoroshiro128+;
- PCG;
- ChaCha20.

**Генератори випадкових чисел Xorshift**, також звані генераторами регістрів зсуву, є класом генераторів псевдовипадкових чисел, які були винайдені Джорджем Марсалією. Вони являють собою підмножину регістрів зсуву з лінійним зворотним зв'язком (LFSR), які дозволяють особливо ефективну реалізацію в програмному забезпеченні без надмірного використання розріджених поліномів. Вони генерують наступне число у своїй послідовності, неодноразово беручи ексклюзивне число або його версію з бітовим зсувом. Це робить виконання надзвичайно ефективним на сучасних комп'ютерних архітектурах, але це не впливає на ефективність апаратної реалізації. Як і всі LFSR, параметри потрібно вибирати дуже ретельно, щоб досягти тривалого періоду[14].

Для виконання в програмному забезпеченні генератори xorshift є одними з найшвидших криптографічно не захищених генераторів випадкових чисел, які вимагають дуже маленького коду та стану. Однак вони не проходять усі статистичні перевірки без подальшого уточнення. Ця слабкість виправляється шляхом поєднання їх із нелінійною функцією, як описано в оригінальній статті. Оскільки звичайні генератори xorshift (без нелінійного кроку) не проходять деякі статистичні тести, їх звинувачують у ненадійності.

Xoroshiro128+ (названий на честь своїх операцій: XOR, rotate, shift, rotate) — це генератор псевдовипадкових чисел, призначений як наступник xorshift+. Замість увічнення традицій Marsaglia xorshift як базової операції, xoroshiro128+ використовує лінійне перетворення на основі зсуву/обертання, розроблене Себастьяно Вінья у співпраці з Девідом Блекманом. Результатом є значне покращення швидкості та якості статистики.

**Залежність якості алгоритму від параметрів** – Дуже велика. Алгоритм має гарні характеристики якщо параметри підібрані правильно.

**Кількість початкових даних необхідна для генерації –** Мала. Параметри алгоритму це число початку так зване зерно і три зміні для засувів які мають значення простих чисел. Важливо щоб зерно не будо рівним 0.

**Довжина генеруємого періоду -** $2^{128} - 1$.

**Криптографічна стійкість –** Алгоритм не є криптографчно стійким.

**Програмна реалізація алгоритму –** Алгоритм є дуже легким у реалізації на сучасних системах.

**Навантаження на систему –** Мінімальне. Для роботи потребує збереження у пам’яті змінної стану.

**Швидкість генерації –** Дуже швидкий. На сгоднішній день в вражається най швидшим ПВГЧ.

**Статистична якість –** Гарна. Проходить майже всі існуючі тести на статистичну якість.

**Важкість передбачення –** Можливі передбачення. Попередні версії цього алгоритму мали проблеми з передбачуваністю результатів а більш сучасні версії погано дослідженні в цьому напрямі.

**Відтворення результатів –** Так. Алгоритм має можливість відтворити результати при однакових вхідних параметрах.

**ChaCha20**. Цей RNG надається в системах OpenBSD як заміна arc4random (і дещо заплутано надається під назвою arc4random для зворотної сумісності). Він заснований на потоковому шифрі ChaCha20 Деніела Дж. Бернштейна, який є варіантом його шифру Salsa20[15].

**Залежність якості алгоритму від параметрів –** Серденя. Алгоритм має гарні характеристики при більшості налаштувань.

**Кількість початкових даних необхідна для генерації –** Серденя.

**Довжина генеруємого періоду –** $2^{128}$.

**Кориптографічна стійкість –** Криптографічно стійкий. Даний алгоритм проходить тести на криптографічну стійкість і не вразливій до основних типів атак.

**Програмна реалізація алгоритму –** Дуже важкий у реалізації. Має складний у реалізації алгоритм з великою кількістю станів.

**Навантаження на систему –** Середьне. Для роботи алгоритму потрібно 0,1 КБ пам’яті що у можна в вважати середнім значення серед ГПВЧ.

**Швидкість генерації –** Повільно. Генерує числа дуже повільно.

**Статистична якість –** Задовільна. Проходить більшість тестів на статистичну якість але не всі.

**Важкість передбачення –** Складна. Згідно дослідженням цього алгоритму можна сказати що його маже не можливо передбачити.

**Відтворення результатів –** Так. Алгоритм має можливість відтворити результати при однакових вхідних параметрах.

**Переставлений конгруентний генератор** (PCG) — це алгоритм генерації псевдовипадкових чисел, розроблений у 2014 році, який застосовує вихідну функцію перестановки для покращення статистичних властивостей лінійного конгруентного генератора за модулем 2n. Він досягає чудової статистичної продуктивності завдяки невеликому та швидкому коду та малому розміру стану[16].

PCG відрізняється від класичного лінійного конгруентного генератора (LCG) трьома способами:

модуль LCG і стан більші, як правило, вдвічі більші за бажаний вихід,

він використовує модуль ступеня 2, що забезпечує особливо ефективну реалізацію з генератором повного періоду та незміщеними вихідними бітами, і

стан не виводиться безпосередньо, а скоріше старші біти стану використовуються для вибору побітового повороту або зсуву, який застосовується до стану для створення виводу[17].

Саме обертання змінної усуває проблему короткого періоду в молодших бітах, від якої страждають LCG зі ступенем 2.

**Залежність якості алгоритму від параметрів –** Серденя. Алгоритм має гарні характеристики при більшості налаштувань.

**Кількість початкових даних необхідна для генерації –** Мала. Початковими даними для алгоритму є дві змінних.

**Довжина генеруємого періоду -** $2^{128}$ .

**Кориптографічна стійкість –** Досліджується. Так як алгоритм відносно новий його стійкість до атак вивчається.

**Програмна реалізація алгоритму –** Алгоритм є дуже легким у реалізації на сучасних системах.

**Навантаження на систему –** Мінімальне. Для роботи потребує збереження у пам’яті кількох змінних.

**Швидкість генерації –** Велика. Відносно інших алгоритмів має достатньо велику швидкість.

**Статистична якість –** Дуже гарна. Проходить всі сучасні перевірки на статистичну випадковість.

**Важкість передбачення –** Дуже складна. Згідно з останніми дослідженнями алгоритм важко передбачити але дослідження в цьому напрямку ще йдуть.

**Відтворення результатів –** Так. Алгоритм має можливість відтворити результати при однакових вхідних параметрах.

В таблиці 2.2 Переведено всі алгоритми та їх критерії.

Таблиця 2.2 - Порівння алгоритмів.

| Назва параметру | Xoroshiro128+ | PCG | ChaCha20 |
|---|---|---|---|
| Залежність якості алгоритму від параметрів | Велика | Середня | Середня |
| Кількість початкових даних необхідна для генерації | Мала | Мала | Середня |
| Довжина генерує мого періоду | $2^{128}$-1 | $2^{63}$ | $2^{128}$ |
| Кориптографічна стійкість | Ні | Так | Так |
| Програмна реалізація алгоритму | Дуже легка | Легка | Дуже складна |
| Навантаження на систему | Мінімальне | Мінімальне | Середнє |
| Швидкість генерації | Дуже швидка | Швидка | Середня |

Продовження таблиці 2.2 - Порівння алгоритмів.

| Назва параметру | Xoroshiro128+ | PCG | ChaCha20 |
|---|---|---|---|

| Статистична якість | Дуже гарна | Гарна | Задовільна |
|---|---|---|---|
| Важкість передбачення | Можливі передбачення | Дуже складна | Складна |
| Відтворення результатів | Так | Так | Так |

2.1.3 Обґрунтування вибору алгоритму.

Серед розглянутих алгоритмів Xoroshiro128+ має непогані характеристики але не є криптографічно безпечним що звужує область його застосування окрім цього алгоритм має дуже велику залежність у якості генерованих чисел від налаштувань алгоритму що є критичним при генеруванні чисел на основі нестандартних даних. Тому він не підходить.

ChaCha20 при більшості налаштувань генерує числа з задовільними параметрами але при цьому його важно реалізовувати і він використовує дуже багато ресурсів системи відносно інших алгоритмів. При цьому він є криптографічно безпечним але він не підходить так як інші його характеристики не задовольняють наших вимог.

PCG серед розглянутих алгоритмів має найкращі характеристики майже у кожному пункті при цьому є криптографіччно безпечним. Але головною характеристикою є мала кількість даних що потрібна для генерації чисел та те що при більшості налаштувань алгоритм гарну якість генерованих чисел.

## 2.2 Аналіз даних які необхідні для обраного генератора.

Для модифікації алгоритму потрібно з’ясувати як алгоритм генерує випадкові числа для того щоб зрозуміти які дані та які значення необхідні для генерації. Сімейство генераторів PCG для генерації використовує модифікований LCG який визначається рекурентним співвідношенням(2.1).

$$X_{m+1} = (aX_n + c) \bmod m \qquad (2.1)$$

Де X це - послідовність псевдовипадкових значень послідовність.

m, 0 < m – модуль

a, 0 < a < m – множник

c, 0 ≤ c < m – приріст

$X_0, 0 \leq X_0 < m$ – початкове значення

Як можна побачити для генерації чисел за допомогою LCG нам необхідно чотири змінних які що найменше повинні бути більше нуля. Але для обраної варіації алгоритму а саме PCG-XSH-RR достатньо усього трьох а саме початкового значення або так званого зерна алгоритму, множника та приріст.

Сімейство PCG на відміну від LCG для генрації даних використовує тільки половину бітів від числа стану наприклад якщо число стану 64 біти то генеруєме число буде розміром в 32 біти. Іншою особливістю PCG є те що вони складаються з різних модулів які можуть бути використані для створення алгоритмів з різними властивостями необхідними для конкретної задачі.

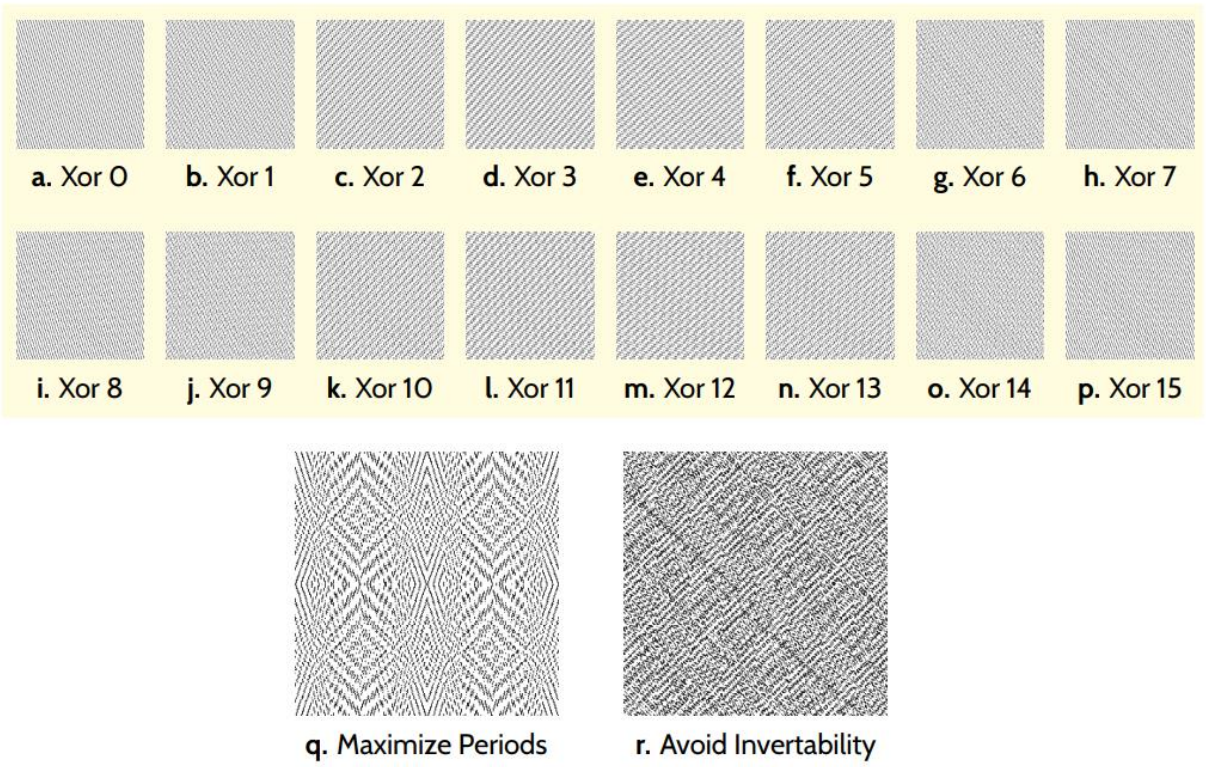
PCG-XSH-RR алгоритм складається із двох модулів а саме XSH та RR. В сімействі алгоритмів PCG є різні модулі з яких складаються різні їх варіації але ми розглянемо тільки ті що нас цікавлять.

XSH ця операція була запозичена ц іншого генератору випадкових чисел та вдосконалена. Генератори випадкових чисел Xorshift, також звані генераторами регістрів зсуву. Вони являють собою підмножину регістрів зсуву з лінійним зворотним зв'язком (LFSR). Вони генерують наступне число у своїй послідовності, неодноразово беручи ексклюзивне або число зі зміщеною бітовою версією. Дану операцію можна представити наступною функцією 2.2:

$$f_c(n) = n \oplus g(c) \tag{2.2}$$

Де g визначають наступним чином: якщо n ∈ $\mathbb{Z}_{2^p}$ та c ∈ $\mathbb{Z}_{2^q}$ тоді g : $\mathbb{Z}_{2^p} \rightarrow \mathbb{Z}_{2^q}$ це можна записати як g(c) = c ◁ (p - q).

На картинці 2.1 можна побачити як Xorshift впливає на дані при застосуванні його до різних бітів.




Малюнок 2.1 Застосування Xorshift

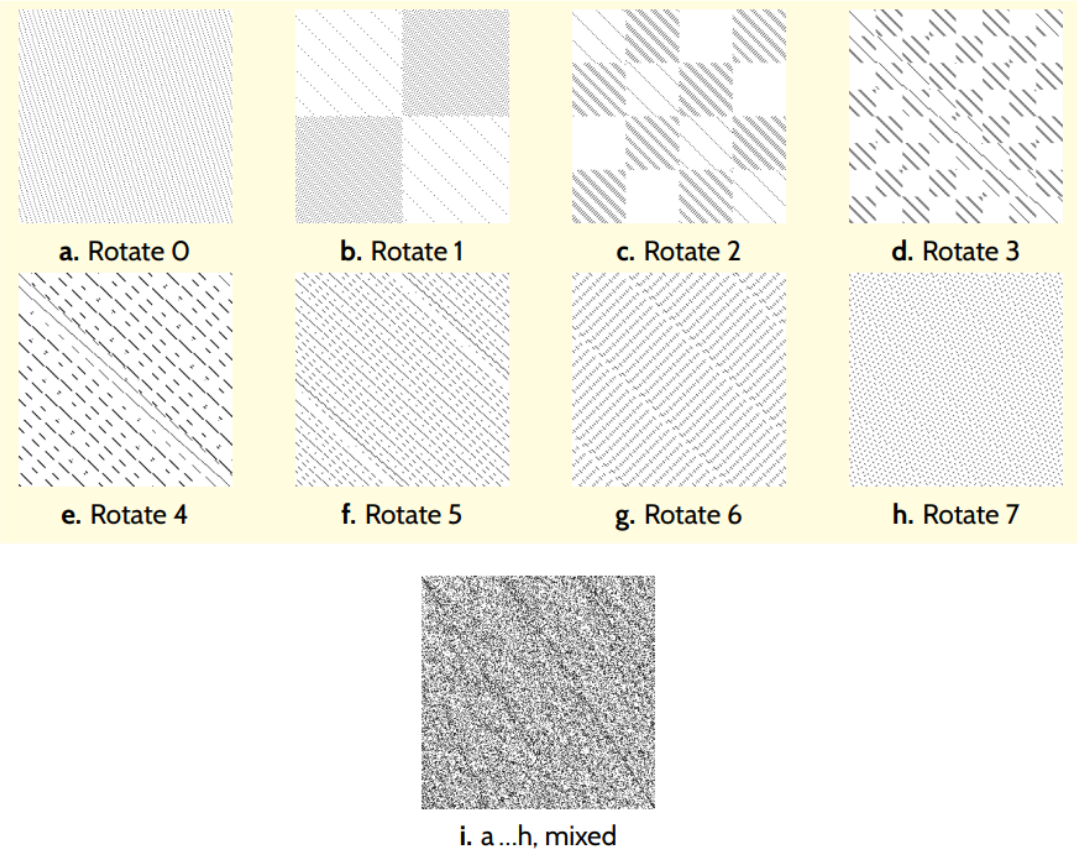
Константа вибирається як половина бітів, які не відкидаються наступною операцією (округляються в меншу сторону).

RR це операція порозрядного повороту. Яку ми застосовуємо до половини бітів це збільшує випадковість за рахунок того що старші біти у стандартному LCG більш випадкові чим молодші і їх переміщення збільшує випадковість усього значення. Дану операцію можна визначити наступною функцією 2.3:

$$f_c(n) = (n \triangleright r) \circlearrowleft c \tag{2.3}$$

Де r це кількість бітів результату

На малюнку 2.2 можна побачити як операція повороту впливає на дані при застосуванні її до різних бітів.




Малюнок 2.2 Застосування операції повороту.

Кількість бітів на які застосовується операція повороту це ще один із параметрів алгоритму який необхідний для генерації.

Усього маємо чотири змінні параметрів алгоритму які необхідні для генерації псевдовипадкових чисел. Зведемо всі необхідні дані в таблицю 2.3 для кращого розуміння яка саме дані необхідні для генерації псевдовипадкових чисел.

Таблиця 2.3 - Дані необхідні для генерації випадкових чисел.

| Назва параметру | Обмеження по значенням | Розмір |
|---|---|---|
| Початкове значення | $0 \neq X$ | 64 b |
| Множник | $0 \neq X\ 1 \neq X$ | 64 b |
| Приріст | $0 \neq X$ | 64 b |
| Кількість бітів повороту | $0<X<32$ | 4 b |

Як видно з таблиці 2.3 для усього для роботи ГПВЧ потрібно 196 bit інформації які не повинні бути рівним нулю.

## 2.3 Вибір даних на основі яких будуть генеруватися випадкові числа

З’ясувавши які дані обраний алгоритм ГПВЧ використовує для генерації можна обрати дані на основі яких будемо генерувати випадкові числа. Як відомо усі дані в цифрованому виді це набір одиниць і нулів які ми можемо використовувати у якості даних для алгоритму. Тому потрібно розробити алгоритм який буде перетворювати потік одиниць та нулів у параметри алгоритму.

## 2.4 Розробка алгоритму перетворення потоку даних у параметри алгоритму

Потрібно розробити алгоритм який на вхід уде приймати потік даних будь-якого розміру та на основі них створювати один або декілька наборів параметрів алгоритму. Також алгоритм повинен перевіряти наскільки отриманий набір відповідає вимогам алгоритму тобто перевіряти щоб параметри не були нульовим. Ще однією вимогою до алгоритму буде видозмінення даних таким чином щоб схожі потоки даних мали не схожі один на одного параметри для того щоб схожі між собою потоки даних генерували принципово різні набори випадкових чисел.

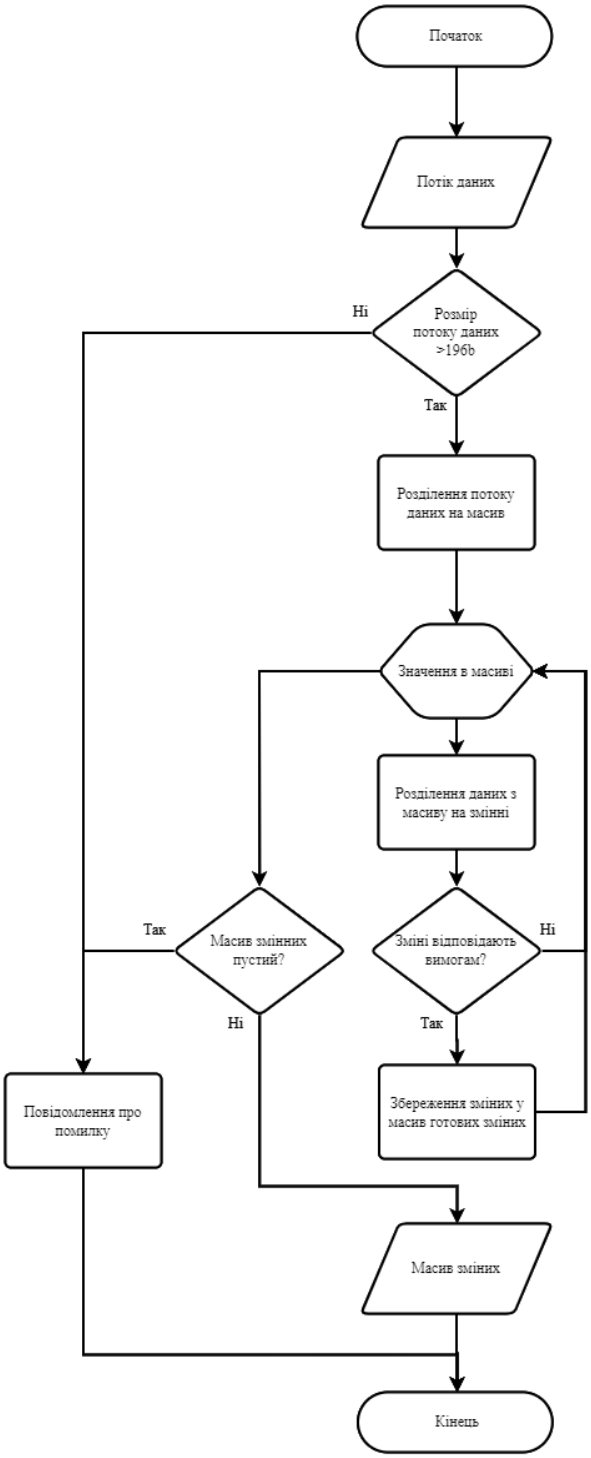


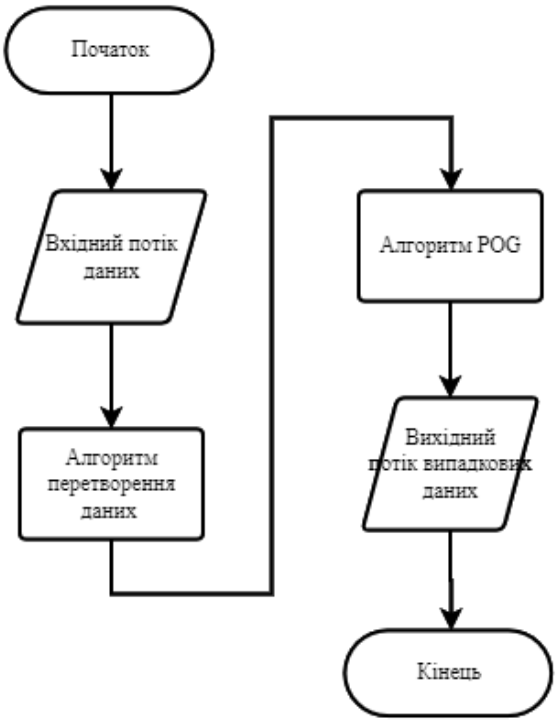

Рисунок 2.1 Алгоритм перетворення потоку даних на зміні до алгоритму

Отже спочатку алгоритм повинен перевірити наскільки великий набір даних було отримано і якщо він менше а ніж нам потрібно то він повинен повідомити про помилку. Якщо набір даних відповідає вимогам мінімального розміру то ми розділяємо данні по міняльним блокам а саме 196b після чого до отриманих блоків

застосовуємо функцію хешування далі отримані дані ми ділимо на параметри змінних алгоритму тобто на три змінних по 64b і одну 4b далі перевіряємо чи підходять нам значення змінних якщо підходять то зберігаємо у масив з параметрами і переходимо до наступного блоку даних якщо ні то просто переходимо до наступного блоку даних якщо блоки даних скінчилась ми перевіряємо чи є хоча б один набір параметрів алгоритму якщо так виводимо його і завершуємо алгоритм якщо ні то виводимо помилку і завершуємо алгоритм.

## 2.5 Принцип роботи модифікованого алгоритму генерації випадкових чисел

Модифікований алгоритм можна побачити на малюнку 2.2 на вхід алгоритму подаються дані у вигляді двійкового коду а на виході алгоритму ми отримуємо випадкові числа також у вигляді двійкового коду. Після отримання двійкових даних алгоритм використовує раніше розроблений метод перетворення даних у параметри алгоритму після чого набір параметрів алгоритму передається до стандартного алгоритму генерації. Алгоритм ГПВЧ генерує випадкові дані які і повертає алгоритм.




Малюнок 2.2 - Модифікований алгоритм ГПВЧ

Важливо щоб дані які потратлять в алгоритм містили хоча б 196b інформації якщо алгоритм отримає менше даних він не зможе згенерувати жодного набору параметрів до алгоритму та повідомить про помилку якщо це станеться генерації випадкових чисел не відбудеться і алгоритм замість випадкового потоку даних поверне  пустий потік даних.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 3.1 Створення бібліотеки для генерації чисел

Для перевірки працездатності алгоритму було вирішено написати бібліотеку яка буде містити в собі програму реалізацію генератора випадкових чисел на основі нестандартних даних.

### 3.1.1 Обґрунтування вибору мови програмування.

Бібліотека написана на мові програмування C#.

C# - це багато парадигмальна мова програмування загального призначення високого рівня. C# охоплює статичну типізацію, сильну типізацію, лексичну, імперативну, декларативну, функціональну, загальну, об’єктно-орієнтовану (на основі класу) та компонентно-орієнтовану дисципліни програмування.

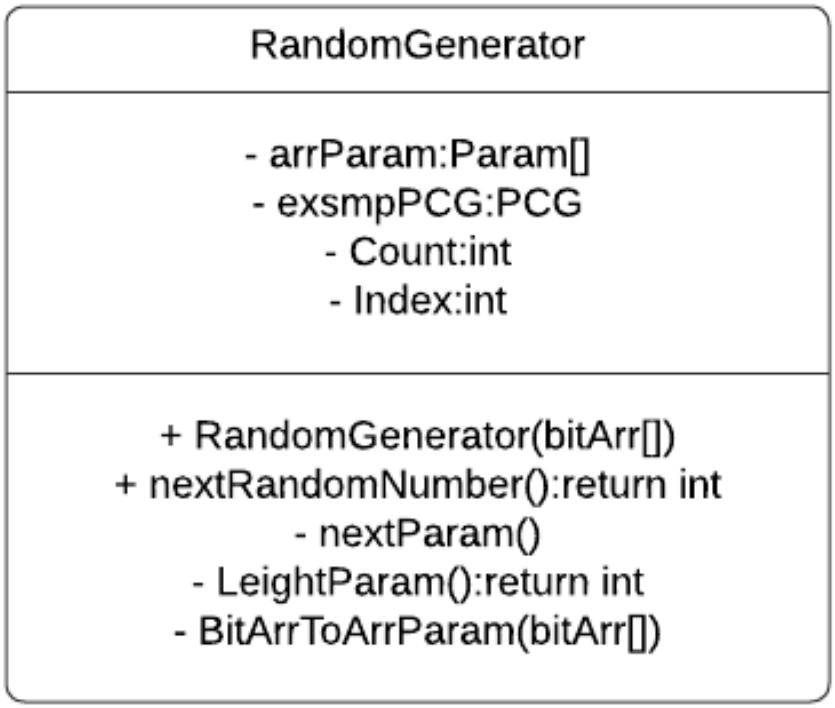
Мова програмування C# була розроблена Андерсом Хейлсбергом із Microsoft у 2000 році та пізніше була затверджена як міжнародний стандарт Ecma (ECMA-334) у 2002 році та ISO/IEC (ISO/IEC 23270) у 2003 році. Microsoft представила C# разом із .NET Framework і Visual Studio, обидва з яких були закритими. У той час у Microsoft не було продуктів з відкритим кодом. Через чотири роки, у 2004 році, розпочався безкоштовний проект з відкритим вихідним кодом під назвою Mono, що надає кросплатформний компілятор і середовище виконання для мови програмування C#. Через десять років Microsoft випустила Visual Studio Code (редактор коду), Roslyn (компілятор) і уніфіковану платформу .NET (фреймворк програмного забезпечення), які підтримують C# і є безкоштовними, з відкритим кодом і кросплатформенними. Mono також приєднався до Microsoft, але не був об’єднаний із .NET.

### 3.1.2 Структура бібліотеки

Бібліотека містить публічний головний клас який і буде використовуватися для генерації випадкових чисел який називається RandomGenerator. А також допоміжний клас який використовується в середні головного з назвою PCG.

#### 3.1.2.1 Клас RandomGenerator.

Клас завдання якого генерувати числа на основі вхідного протоку двійкових даних. Структура класу зображена на Малюнку3.1.



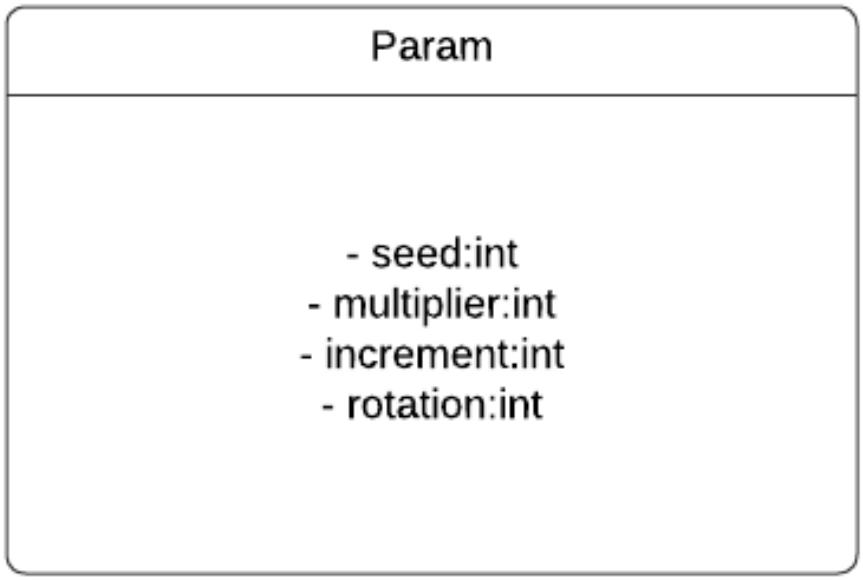

Малюнок 3.1 – Структура класу RandomGenerator

Клас містить чотири змінних:

- arrParam;
- exsmpPCG;
- Count;
- Index.

Змінна exsmpPCG Зберігає екземпляр класу PCG за допомогою якого генеруються числа. Змінна Count зберігає кількість генерованих чисел алгоритмом. Index – Зберігає індекс параметрів алгоритму які використовуються для генерації.

Зміна arrParam зберігає масив параметрів за допомогою яких буде виконуватися генерація. Цей масив зберігає дані у вигляді структури яка зображена на малюнку 3.2.




Малюнок 3.2 – Структура даних Param

Структура містить три змінних:

- Seed – Зерно алгоритму тип даних int;
- Multiplier – Множник алгоритму тип даних int;
- Increment – Приріст алгоритму тип даних int;
- Rotation – Кількість бітів на які буде відбуватися поворот тип даних int.

Клас має наступний список методів:

- RandomGenerator;
- nextRandomNumber;
- nextParam;

- LeightParam;
- BitArrToArrParam.

Метод RandomGenerator це публічний метод конструктор класу він має один параметр це сам набір даних який буде використовуватися для генерації випадкових чисел. При виконані цього методу викорчується метод BitArrToArrParam а також створюється екземпляр класу PCG з першим набором параметрів.

Метод nextRandomNumber це публічний метод який не має ніяких параметрів та повертає випадкове число типу int. Метод використовується для отримання випадкових чисел. Даний метод перевіряє скільки чисел згенеровано за допомогою даних обраних параметрів якщо кількість випадкових чисел які може згенерувати вичерпалася визивається метод nextParam після чого звертається до екземпляру класу PCG і визиває два методи nextState та ReturnNumber.

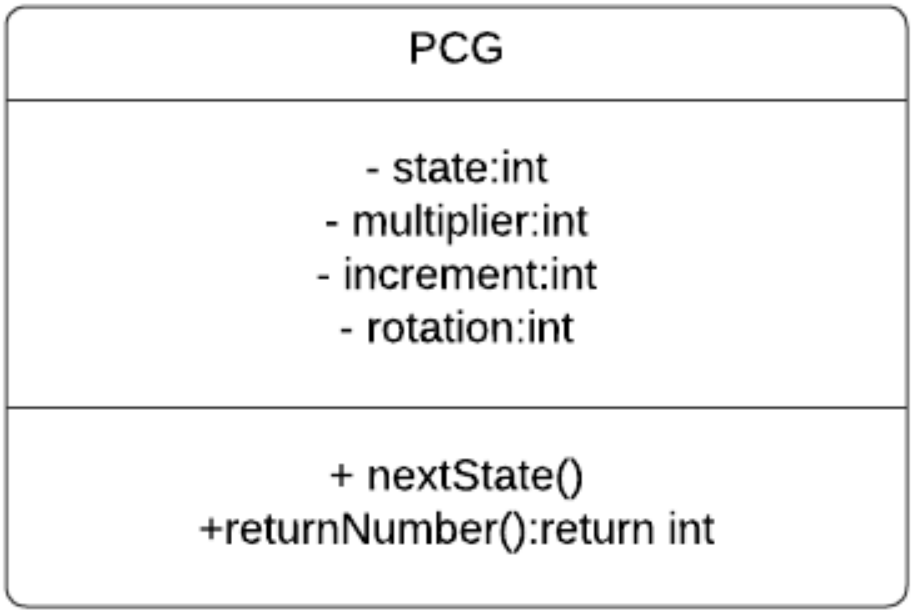
Метод nextParam публічний метод який не має ніяких параметрів та не повертає ніяких даних. Цей метод використовується для зміни параметрів алгоритму PCG. При виконані цього методу створюється новий екземпляр класу PCG а також обтуляться кількість згенерованих чисел та збільшується на одиницю номер наступного параметру.

Метод LeightParam публічний метод який не має ніяких параметрів повертає кількість параметрів які ще може використовувати алгоритм для генерації тип даних int.

Метод BitArrToArrParam приватний метод який приймає у якості параметру потік двійкових даних та нічого не повертає. Алгоритм роботи методу можна побачити на малюнку 2.1.

#### 3.1.2.1 Клас PCG.

Клас в якому реалізовано метод генерації випадкових чисел структура класу зображена на малюнку 3.3.




Малюнок 3.3 – Структура класу PCG

Клас містить чотири змінних а саме:

- state – Внутрішній стан алгоритму тип даних int;
- multiplier – Множник алгоритму тип даних int;
- increment – Приріст алгоритму тип даних int;
- rotation – Кількість бітів на які буде відбуватися поворот тип даних int.

До методів класу відносяться два методи метод nextState який виконує перетворення в середні класу та returnNumber який перетворює внутрішній стан алгоритму на число результату.

### 3.2 Перевірка статистичних властивостей алгоритму

Для перевірки чи насправді алгоритм генерує числа які мають властивості випадкових чисел було вирішено проти ряд статистичних тестів а саме TestU01.

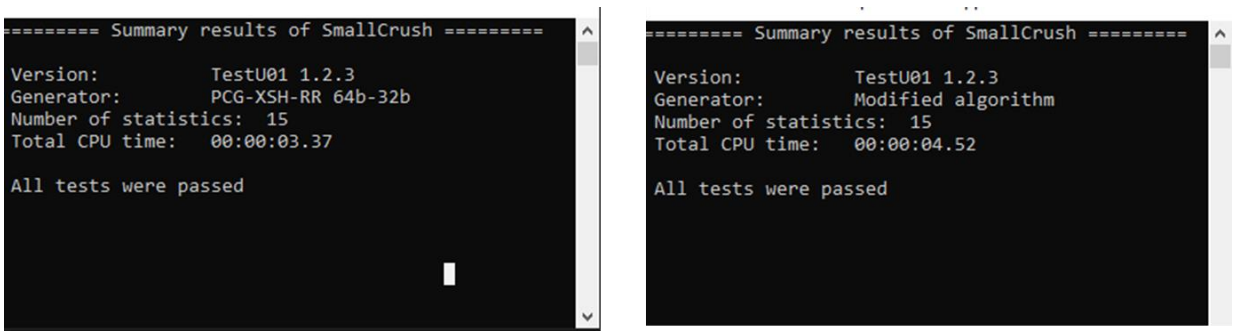
TestU01 — це бібліотека програмного забезпечення, реалізована мовою ANSI C, яка пропонує набір утиліт для емпіричного тестування на випадковість

генераторів випадкових чисел. Бібліотека була вперше представлена в 2007 році П’єром Л’Екюєром і Річардом Сімардом з Університету Монреаля[18].

Бібліотека реалізує кілька типів генераторів випадкових чисел, включаючи деякі запропоновані в літературі та деякі з широко використовуваних програм. Він забезпечує загальну реалізацію класичних статистичних тестів для генераторів випадкових чисел, а також кілька інших, запропонованих у літературі, і деякі оригінальні. Ці тести можна застосувати до генераторів, попередньо визначених у бібліотеці, генераторів, визначених користувачем, і потоків випадкових чисел, що зберігаються у файлах. Також доступні спеціальні набори тестів для послідовностей уніфікованих випадкових чисел у [0,1] або послідовностей бітів. Також надаються базові інструменти для побудови векторів точок, створених генераторами.

TESTU01 пропонує кілька наборів тестів, включаючи «Small Crush» (який складається з 10 тестів), «Crush» (96 тестів) і «Big Crush» (106 тестів). Конкретні тести, що застосовуються для кожної батареї, детально описані в посібнику користувача. На 1,7 ГГц Pentium 4 під керуванням Red Hat Linux 9.0 для простого RNG Small Crush займає приблизно 2 хвилини. Дроблення займає приблизно 1,7 години. Big Crush займає близько 4 годин[19].

Для перевірки було обрано набір тестів під назвою «Small Crush» результати проходження цього тесту звичайним алгоритмом та його модифікованою версією приведено на малюнку 3.4

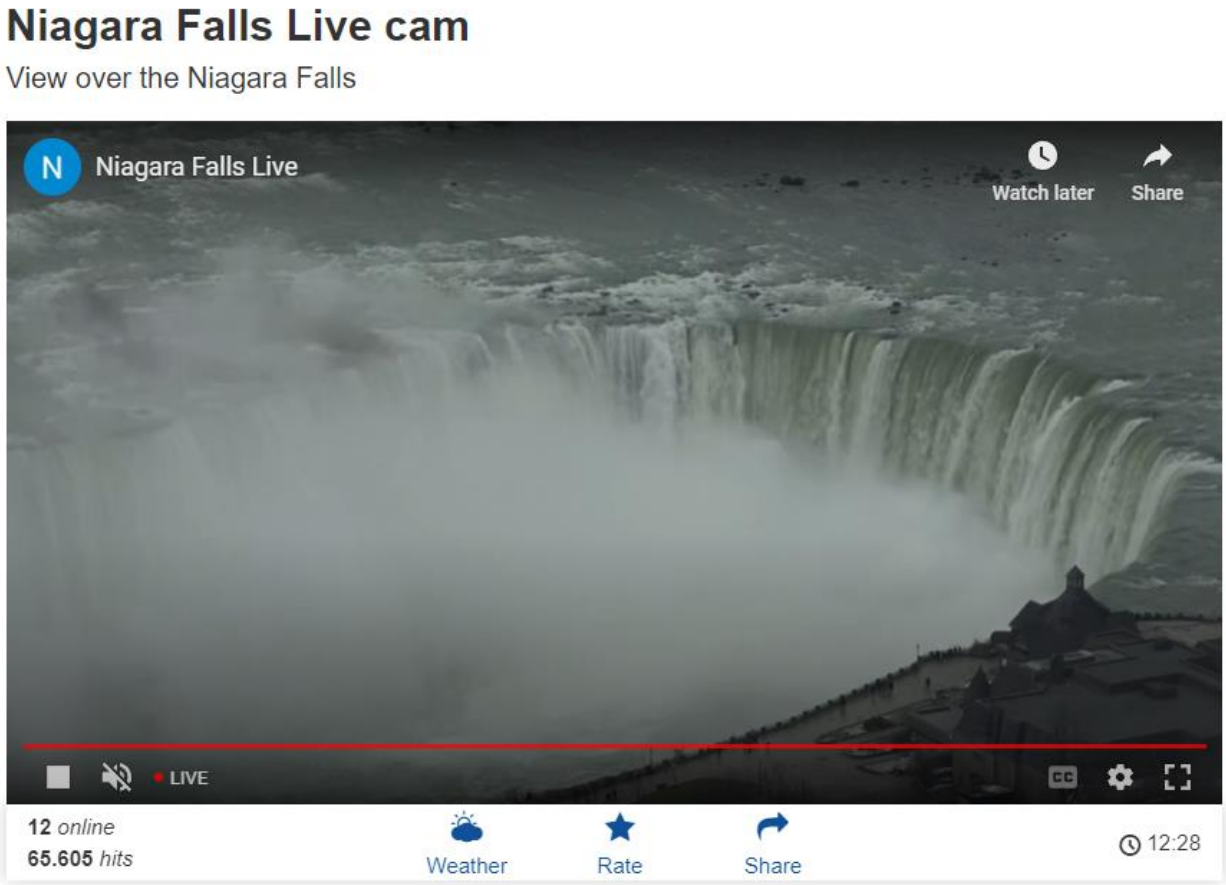



Малюнок 3.4 – Результати тестів

Як можна побачити різниці може розходженням цих тестів у модифікованого алгоритму та стандартної його версії майже немає відрізняється тільки час проходження.

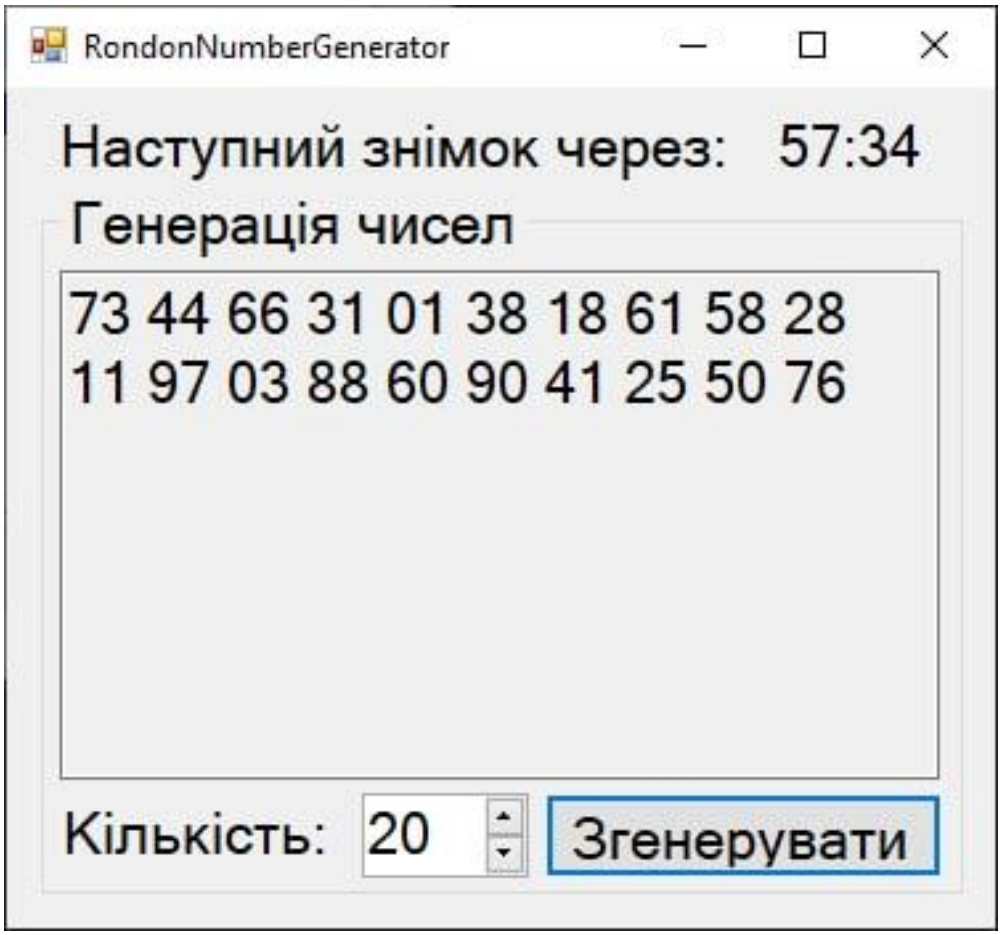
### 3.3 Генерація випадкових чисел на основі нестандартних даних

Для демонстрації можливостей модифікованого алгоритму створено програму яка генерує випадкові числа. У якості даних на основі яких генеруються випадкові числа було обрано онлайн стрім Ніагарського водоспаду. На сайті https://www.skylinewebcams.com/ru/webcam/canada/ontario/niagara-falls/niagara-falls.html йде стрім з камери яка направлена на водоспад скріншот сайту зображено на малюнку 3.4. Даний стрім можна в вважати природнім джерелом випадковості так як доволі складно передбачити як саме поведе себе водоспад.




Малюнок 3.4 – Скріншот сайту з трансляцією

Програма раз в годину отримує скріншот Ніагарського водоспаду перетворює його на послідовність одиниць на нулів після чого передає цю послідовність у клас бібліотеки який займається генерацією випадкових чисел. Далі бібліотека формує список параметрів до алгоритму якщо зі скріншота не вийшло згенерувати жодного набору параметрів програма знову отримує інший скріншот. Коли алгоритм зміг згенерувати необхідний набір параметрів програма починає роботу. Користувачу пропонується обрати кількість випадкових значень які йому потрібно після чого програма генерує необхідну кількість і виводить її користувачу. Користувач знову може запросити набір випадкових чисел до тих пір поки у алгоритму не закінчяться параметри або поки не пройде година після чого програма запросить інший скріншот з сайту і продовжить роботу. На малюнку 3.5 зображено інтерфейс програми.




Малюнок 3.5 – Інтерфейс програми

За рахунок того що дані які отримує ця програма мають природне джерело випадковості статистичні характеристики генерованих чисел наближені до апаратних генераторів. Але дану програму не варто використовувати для цілей криптографії так як вона дуже вразлива за рахунок того що сайт з якого отримуються дані знаходиться у вільному доступі і злочинець може використати це у своїх цілях.

Дана програма не може відтворити значення які вона генерує так як скріншоти на основі яких генеруються дані не зберігаються.

### 3.4 Порівняння характеристик алгоритму при різних даних для генерації

Для того щоб з’ясувати як різні типи вхідних даних впливають на характеристики алгоритму проведемо порівняння різних варіантів алгоритму. Порівнювати будемо варіант використання алгоритму коли на вхід ми отримуємо дані які мають природне джерело випадковості як у демонстраційній програмі з варіантом коли дані взяті з якогось статично джерела наприклад файлу. Також у порівняні візьме участь не модифікована версія алгоритму.

Порівняння алгоритмів буде відбуватися за наступними критеріями:

- Розмір періоду генерації;
- Природне джерело випадковості;
- Швидкість роботи;
- Можливість відтворення результатів;
- Можливість використання в криптографії;
- Статистичні якості.

Стандартний алгоритм представлений у порівняні використовує налаштування які запропонував розробник як ті що дають найкращий статистичний результат. Модифікація що використовує у дані з природнім джерелом випадковості це програма що використовується для демонстрації можливостей

алгоритму. Дані для алгоритму без природньої випадковості це звичайна картинка. Параметри обраних варіантів алгоритмів представлені у таблиці 3.1.

Таблиця 3.1 – Порівняння рівнинних варіантів алгоритму.

| **Назва параметру** | **PCG-XSH-RR 64b-32b** | **Демонстраційна програма** | **Генерація на основі файлів** |
|---|---|---|---|
| Розмір періоду генерації | $2^{63}$ | Мінімальний $-$ $2^{63}$ | Мінімальний $-$ $2^{63}$ |
| Природне джерело випадковості | Ні | Так | Ні |
| Швидкість роботи | Швидкий | Повільний | Повільний |
| Можливість відтворення результатів | Так | Ні | Так |
| Можливість використання в криптографії | Вивчається | Не бажано | Вивчається |
| Статистичні якості | Як у ПВГЧ | Як у апаратного | Як у ПВГЧ |

Як можна побачити з таблиці в залежності від даних які отримує алгоритм його характеристики можуть бути схожими на апаратний генератор випадкових чисел або на програмний генератори випадкових чисел.

# ВИСНОВКИ

Мета роботи створення алгоритму генерації випадкових чисел який для генерації випадкових чисел використовує не стандартні дані досягнута. В роботі:

1. Проаналізовано предметну область ГПВЧ.
2. Проведено аналіз методів генерації випадкових чисел.
3. Проведено порівняльний аналіз існуючих алгоритмів ГПВЧ.
4. Проаналізовано роботу обраного алгоритму ГПВЧ.
5. Модифіковано алгоритм генерації випадкових чисел..
6. Розроблено бібліотеку для генерації випадкових чисел.
7. Розроблено програму для демонстрації можливостей генерування випадкових чисел на основі нестандартних даних.

## СПИСОК ВИКОРСТАНИХ ДЕРЕЛ


1. Philip J. D. Mathematics And Common Sense A Case of Creative Tension / Davis J. Philip.. – 242 с.
2. Карташов М. В. / Імовірність, процеси, статистика. – Київ: ВПЦ Київський університет, 2008.
3. Барковський В. В. Теорія ймовірностей та математична статистика. 5-те видання. — Київ: Центр учбової літератури, 2010. — 424 с.
4. Каленюк П. І. та ін. Теорія ймовірностей і математична статистика. — Львів: Видавництво Національного університету «Львівська політехніка», 2005. — 240 с.
5. L'Ecuyer P. History of uniform random number generation. / Pierre. L'Ecuyer. // Winter Simulation Conference. – 2017.
6. ERNIE — генератор випадкових чисел [Електронний ресурс] – Режим доступу до ресурсу: https://www.i-programmer.info/history/machines/6317-ernie-a-random-number-generator.html.
7. Pironio, S., Acín, A., Massar, та інш. / Random numbers certified by Bell’s theorem.. // Nature. – 2010. – №464. – С. 1021–1024.
8. New quantum method generates really random numbers. // National Institute of Standards and Technology. – 2018.
9. Лічильник Гейгера [Електронний ресурс] – Режим доступу до ресурсу: https://ratingservices.ru/lighting/scetciki-gejgera-princip-raboty.html.
10. All-optical fast random number generator. // Opt. Express. – 2010. – №18. – С. 20360–20369.
11. Klein A. Linear Feedback Shift Registers. / A. Klein. // Stream Ciphers. – 2013. – С. 17–58.
12. Matsumoto M. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator / M. Matsumoto, T. Nishimura. // ACM Trans. Model. Comput. Simul.. – 1998. – С. 3–30.

13. Sebastiano V. An Experimental Exploration of Marsaglia's Xorshift Generators, Scrambled / Vigna Sebastiano. // ACM Trans. Math. Softw.. – 2016. – №4.
14. Marsaglia G. Xorshift RNGs. Journal of Statistical Software / G. Marsaglia. // Journal of Statistical Software. – 2003. – №8. – С. 1–6.
15. Daniel B. ChaCha, a variant of Salsa20. / Bernstein Daniel. – 2008.
16. Melissa E. O'Neill / PCG : A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. – 2014.
17. PCG, A Family of Better Random Number Generators [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.pcg-random.org/index.html.
18. L'Ecuyer P. TestU01: A C Library for Empirical Testing of Random Number Generators / P. L'Ecuyer, S. Richard. // Association for Computing Machinery. – 2007. – №4.
19. TestU01 [Електронний ресурс] – Режим доступу до ресурсу: http://simul.iro.umontreal.ca/testu01/tu01.html.
20. Patrick G. Cyclic Redundancy Check Computation: An Implementation Using the TMS320C54x / Geremia Patrick. – 1999.

# ДОДАТКОК А

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

**МАГІСТЕРСЬКА РОБОТА**

**«ЗАСТОСУВАННЯ МЕТОДУ ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ОСНОВІ НЕСТАНДАРТНИХ ВХІДНИХ ДАНИХ»**

Виконав: студент групи ПДМ-61 Візер Антоній миколайович

Керівник: к.т.н., доц. Трінтіна Наталія Альбертівна

Київ - 2022

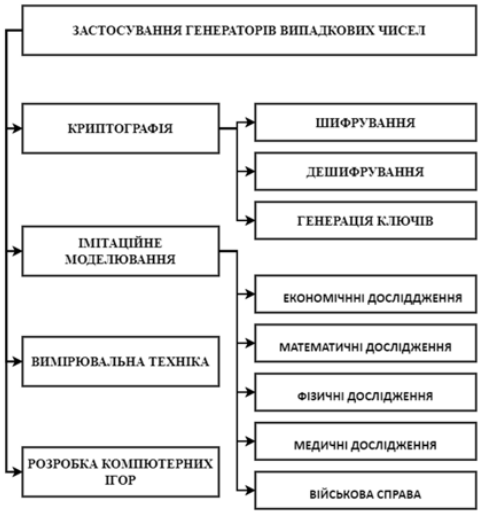
## МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ



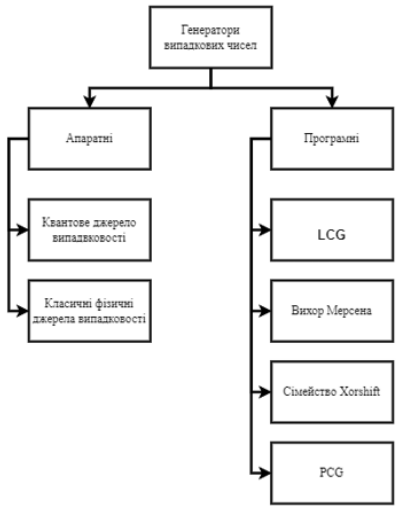
**Мета дослідження**: модифікація алгоритму для збільшення кількості можливих галузей застосування.
**Об'єкт дослідження:** Генерація випадкових даних.

**Предмет дослідження**: Методи генераціїї псевдовипадкових чисел.

# АКТУАЛЬНІСТЬ РОБОТИ



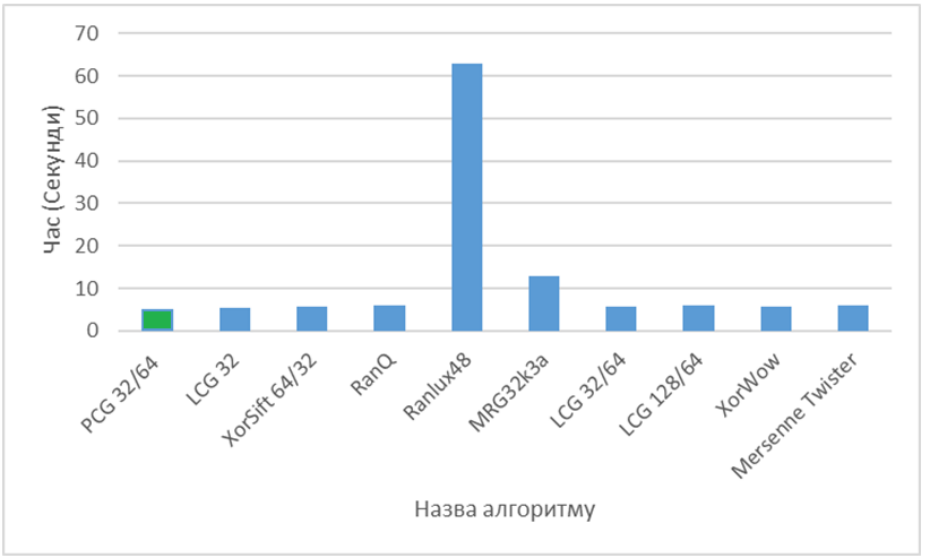

Галузі застосування випадкових чисел




Існуючі алгоритми генерації



# ВИБІР АЛГОРИТМУ ГПВР




Графік швидкості алгоритмів

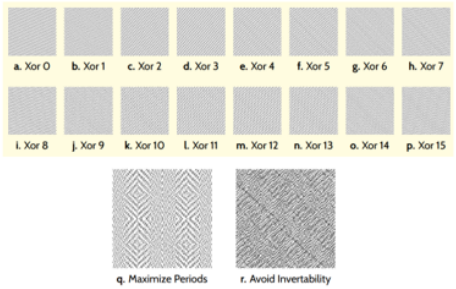
# АНАЛІЗ АЛГОРИТМУ PCG-XSH-RR 64b-32b

**64b-32b**
**64b – кількість біт стану**
**32b – кількість біт результату**

**XSH – Фіксований Xorshifts**

$$f_c(n) = n \oplus g(c),$$

Де n – Віхідне число
c – кількість біт здивигу
g(c) – функція побітного зсуву на c біт

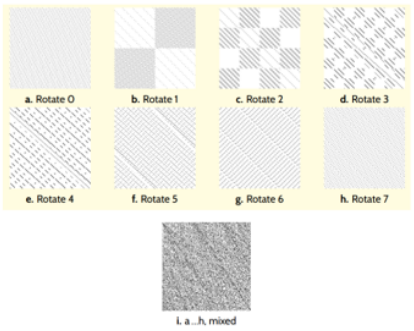



Засстосування Xorshifts
з різними параметрами

**RR – По бітний поворот**

$$f_c(n) = (n \boxtimes r) \circlearrowright c,$$

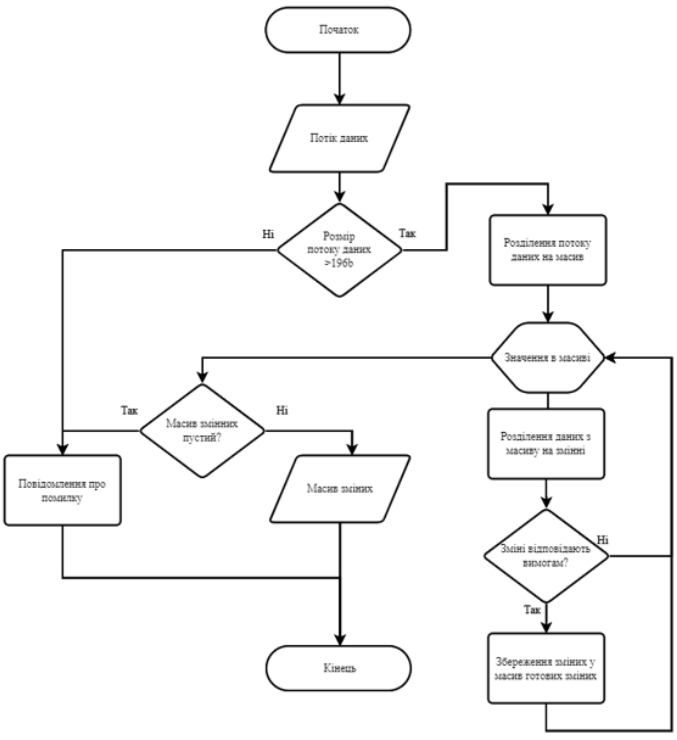
Де n – Віхідне число
r – кількість біт результату
c – кількість біт повороту




Засстосування побітного повороту
з різними параметрами



# АЛГОРИТМ ПЕРЕТВОРЕННЯ ДАНИХ В ПАРМЕТРИ



## ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДИФІКОВАГО АЛГОРИТМУ З СТАНДАРТНИМ

| Назва параметру | Стандартна реалізація PCG-XSH-RR 64b-32b | Модифікований алгоритм на основі природних даних | Генерація на основі файлів |
|---|---|---|---|
| Розмір періоду генераціїї | $2^{63}$ | Залежить від кількості даних більше $2^{63}$ | Залежить від кількості даних більше $2^{63}$ |
| Природне джерело випадковості | Ні | Так | Ні |
| Можливість відтворення результатів | Так | Ні | Так |
| Статистичні якості | Як у ПВГЧ | Як у апаратного | Як у ПВГЧ |
| Можливість використання в криптографії | Вивчається | Можливо | Можливо |
| Вимірювальна техніка | Можливо | Можливо | Можливо |
| Імітаційне моделювання | Не всі дослідження | Можливо | Не всі дослідження |
| Розробка ігор | Можливо | Можливо | Можливо |



## ВИСНОВКИ

1. Проаналізовано предметну область ГПВЧ.
2. Проведено порівняльний аналіз існуючих алгоритмів.
3. Проаналізовано роботу обраного алгоритму.
4. Модифіковано алгоритм генерації випадкових чисел.
5. Розроблено бібліотеку для генерації випадкових чисел.
6. Розроблено програму для демонстрації можливостей генерування випадкових чисел на основі нестандартних даних.
7. Встановлено що, модифікований алгоритм має різні властивості в залежності від даних, які використовуються і дозволяє використовувати їх у більшій кількості галузей.

## ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

**Тези доповідей:**

1. Трінтіна Н.А., Візер А.М. Застосування генераторів псевдо випадкових чисел в сучасному світі // XV НАУКОВО-ТЕХНІЧНІЙ КОНФЕРЕНЦІЇ «СУЧАСНІ ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ» – Київ: ДУТ, 2022.

# ДЯКУЮ ЗА УВАГУ!