Одеський національний університет імені І.І.Мечникова

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему: "Інверсний генератор псевдовипадкових чисел"

(назва українською)

"Pseudo-Random numbers Generators"

(назва англійською)

Виконав:студент денної форми навчання

спеціальності 123 Комп'ютерна інженерія .

(шифр і назва напряму підготовки, спеціальності)

Борисов Демид Сергійович

(прізвище, ім'я, по-батькові)

Керівник Варбанець П.Д.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент Савастру О.В.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2022 р.

Завідувач кафедри

_____ О.В. Савастру (підпис)

(прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «_ _» ____ 2022 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Н.Ф. Казакова

(підпис) (прізвище,ініціали)

Одеса - 2022

АНОТАЦІЯ

У даній дипломній роботі розглядається тема “Генератори псевдовипадкових чисел”.

Мета цієї роботи – це вивчення структури генераторів , архітектуру та програмну реалізацію алгоритмів генерації псевдовипадкових чисел.

У даній дипломній роботі описані основні теоретичні відомості про методи генерування псевдовипадкових чисел а також їх перевірка. Тут будуть розглянуті такі методи , як інверсний конгруетний метод та лінійний конгруетний метод.

Програма виконує усі поставлені задачі над функціями , реалізують генерацію псевдовипадкових чисел та автоматизує процес генерування.

ABSTRACT

In this thesis work, the topic “Pseudo-Random Number Generators” is considered. The metaphor of robotics is the development of the structure of generators, architecture and software implementation of algorithms in the generation of pseudo-reciprocal numbers.

This thesis robot describes the main theoretical statements about the methods of generating pseudo-reciprocal numbers, as well as their re-verification. Here we will look at such methods as the inverse congruent method and the linear congruent method.

The program complete all setting tasks over functions, realizing the generation of pseudo-reciprocal numbers and automating the generation process

ЗМІСТ

ВСТУП

В інформатиці та обчислювальній техніці випадкові числа знаходять широке застосування: для проведення статистичних випробувань, у криптографії , в імітаційному моделюванні . Однак отримання істинно випадкових чисел є вкрай трудомістким процесом. Це викликано в першу чергу складністю і дорожнечею генераторів істинно випадкових чисел. Також слід врахувати, що генерація істинно випадкових чисел може займати багато часу , і цілком імовірний варіант , коли при вичерпанні джерела істинно випадкових чисел програма переходить в режим очікування (час якого також випадкове) . Таким чином , частіше мова йде не про істинно випадкові числа , а про псевдовипадкові числа. У цій роботі ми будемо розглядати програмні реалізації тільки генераторів псевдовипадкових чисел.

Генератори псевдовипадкових чисел повинні задовольняти наступні критерії:

- алгоритм частки;
- для запобігання зациклювання послідовності псевдовипадкових чисел необхідно мати досить довгий період;
- алгоритм повинен бути ефективним за швидкістю роботи алгоритму і витраті обчислювальних ресурсів;
- алгоритм повинен задовольняти критерію відтворюваності, тобто повинна бути можливість відтворити раніше згенеровану; послідовність псевдовипадкових чисел будь-яку кількість разів;
- алгоритм повинен бути переносимий по відношенню до архітектур обладнання та операційних оточень;
- алгоритм повинен бути швидким і ресурсозберігаючим;

Тестування генераторів псевдовипадкових чисел фактично полягає в перевірці послідовності псевдовипадкових величин на незалежність , однакову розподіленість, рівномірність на одиничному інтервалі.

В основі кожної програмної реалізації генератора псевдовипадкових чисел лежить свій алгоритм. У багатьох теоретичних дослідженнях обговорюються і порівнюються самі алгоритми , але ми залишимо даний аспект без обговорення. У роботі зосередимося на питаннях практичної перевірки якості генерованих програмним кодом послідовностей псевдовипадкових чисел. Дослідження проводиться за допомогою програмних інструментів.

У рамках роботи зачіпаються наступні проблеми . По-перше , у вільних системах комп’ютерної алгебри часто використовують генератори псевдовипадкових чисел, які не пройшли всіх можливих тестів. По-друге , бажано використовувати найкращий генератор псевдовипадкових чисел. Для вирішення цих проблем нами описується практична структура стенду для дослідження програмних реалізацій генераторів псевдовипадкових чисел. Це дозволяє оцінити поточну реалізацію генератора , вбудовану в ту чи іншу систему комп’ютерної алгебри. Крім того , можна підібрати іншу програмну реалізацію для вбудовування її у вільну систему комп’ютерної алгебри .

1 АНАЛІЗ МЕТОДІВ ГЕНЕРУВАННЯ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ

1.1 Генерування рівномірно розподілених випадкових чисел

Рівномірні закони є самими простими для виконання . Як правило, для генерування псевдовипадкових чисел за всіма іншими статистичними законами як заданий генератор використовується закон одиниці. У реалізації використовується випадкова послідовність, сформована спеціальним генератором шуму. У комп'ютерах більш поширені програмні методи генерування випадкових чисел, які обчислюються за формулами, тому вони в принципі не можуть бути випадковими. Ці числа називаються псевдовипадковими числами. Припускаючи, що згенеровані випадкові числа можна використовувати як випадкові числа, якщо вони обчислюються за спеціальною формулою і ведуть себе як випадкові числа в спеціальних тестах. Різниця між псевдовипадковими числами і випадковими числами полягає в тому, що вони були періодичними, тобто повторювали одне і те ж випадкове число з певного періоду часу, що природно, оскільки вони обчислюються за формулою.

Найпоширенішими є наступні методи генерації:

- Метод добутку
- Метод квадратів
- Метод перемішування
- Змішаний конгруентний метод

Метод квадратів .

Існують чотиризначні числа R0. Це число зводиться в квадрат і вноситься в R1. Потім візьміть середнє (середнє з чотирьох чисел) з R1 - нове випадкове число - і запишіть його в R0. Потім процес повторюється (див. Рисунок 1.1). Зауважте, що фактично як випадкове число, не ghij, а 0.ghij - призначає нулі та десяткову крапку зліва. Цей факт відображений на рисунку 1.1 і подібні

цифри, що наведені далі. Цей факт відбитий як на рис. 1.1, так і на подальших подібних рисунках.

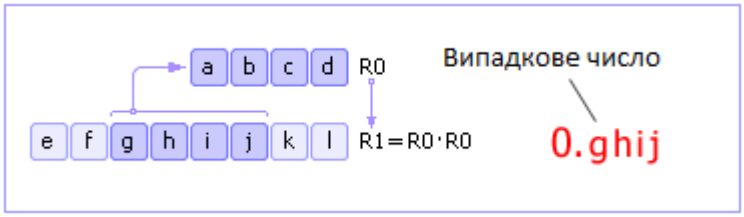

Рис 1.1 – Метод квадратів.

Недоліки цього методу:

1) якщо на певній ітерації значення R0 дорівнює 0, генератор вироджений, тому важливо вибрати правильне початкове значення R0;

2) генератор буде повторювати послідовність через Mn кроків (максимум), де n — число R0, а M — основа системи числення. Якщо число R0 представлено в двійковій системі числення, послідовність псевдовипадкових чисел повториться через $2^4 = 16$ кроків. Зверніть увагу, що послідовність може бути повторена раніше, якщо початковий номер не вибрано вдало. Описаний вище спосіб був запропонований Джоном фон Нейманом і відноситься до 1946 року. Оскільки цей спосіб виявився ненадійним, від нього дуже швидко відмовилися.

Метод перемішування.

У методі перемішування використовуються операції, які повертають вміст комірки ліворуч і праворуч. Ідея методу полягає в наступному. Нехай початкове число R0 зберігається в комірці. Циклічно зміщуючи вміст комірки вліво на 1/4 довжини комірки, отримуємо нове число R0*. Аналогічно, зациклившись вміст комірки R0 на 1/4 довжини комірки вправо, ми отримаємо друге число, R0**. Сума чисел R0* і R0** дає нове випадкове число R1. Потім R1 вводиться в R0, і вся послідовність операцій повторюється (див. рис 1.2).

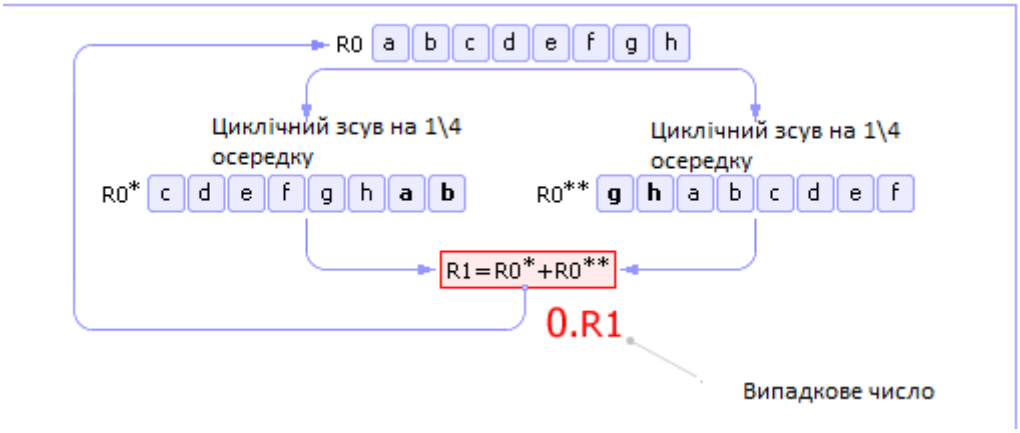

Рис 1.2 – Метод перемішування

Зверніть увагу, що числа, отримані шляхом підсумовування R0* і R0**, можуть не точно вписуватися в клітинку R1. У цьому випадку з отриманого числа необхідно відкинути зайві цифри. Пояснимо це для графіка. 1.2, де всі клітинки представлені вісьмома двійковими цифрами. Нехай R0 * = 100100012 = 14510, R0 ** = 101000012 = 16110, тоді R0 * + R0 ** = 1001100102 = 30610. Як бачимо, число 306 займає 9 цифр (двійкове), а клітинка R1 (як і R0) може містити до 8 цифр. Тому перед введенням значення в R1 з числа 306 потрібно видалити один «зайвий» крайній лівий біт, в результаті чого R1 буде більше не 306, а 001100102 = 5010. Також зауважте, що в таких мовах, як Pascal, який «обрізає» зайві біти при переповненні комірки, це робиться автоматично на основі вказаного типу змінної. Метод добутку.

Число R0 множиться на R1, з отриманого результату R2 витягається середина R2* (це чергове випадкове число) і множиться на R1. За цією схемою обчислюються усі подальші випадкові числа (див. рис. 1.3).

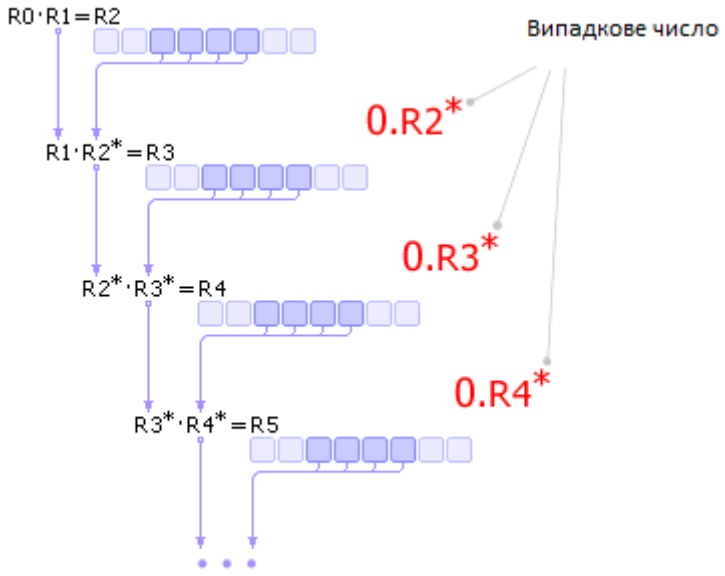

Рис 1.3 – Метод добутку

1.2 Лінейний конгруетний метод

Лінійний конгруетний метод є однією з простих і найбільш споживаних нині процедур , що імітують випадкові числа. У цьому методі використовується операція mod(x,y) , що повертає залишок від ділення першого аргументу на другий. Кожне подальше випадкове число розраховується на основі попереднього випадкового числа по наступній формулі:

$$r_{i+1} = \mathrm{mod}(k \cdot r_i + b, M). \quad (1.1)$$

де - M — модуль ($0 < M$);

k — множник ($0 \le k < M$);

b — приріст ($0 \le b < M$);

r_0 — початкове значення ($0 \le r_0 < M$).

Послідовність випадкових чисел, отримана за цією формулою, називається лінійною конгруентною послідовністю. Багато авторів називають лінійні конгруентні послідовності при b = 0 мультиплікативними конгруентними методами, а лінійні конгруентні послідовності при $b \neq 0$ — гібридними конгруентними методами. Для генератора маси потрібно вибрати відповідні коефіцієнти. Число M

має бути достатньо великим, оскільки період не може мати більше ніж M елементів. З іншого боку, поділ, який використовується в цьому методі, є досить повільною операцією, тому для двійкового комп'ютера логічно вибрати M = 2n, тому що в цьому випадку знаходження залишку від поділу всередині комп'ютера зводиться до двійкової логічної операції " i". Також поширеним є вибір найбільшого простого числа M, меншого за 2n: у спеціальній літературі показано, що в цьому випадку нижчі біти отриманого випадкового числа ri + 1 такі ж випадкові, як і старі, що позитивно впливає на всю послідовність випадкових чисел. Прикладом є одне з чисел Мерсенна, яке дорівнює 2^{31} - 1, тому M = 2^{31} - 1. Однією з вимог до лінійно конгруентних послідовностей є якомога більший період. Тривалість періоду залежить від значень M, k і b. Теорема, яку ми пропонуємо нижче, дозволяє визначити, чи можна досягти періоду максимальної довжини для певних значень M, k і b.

Теорема. Лінійна конгруентна послідовність, визначена числами M, k, b і r0, має період завдовжки M тоді і тільки тоді, коли:

- числа b і M взаємно прості;

- k - 1 кратно p для кожного простого p, що є дільником M;

- k - 1 кратно 4, якщо M кратно 4.

Нарешті, ми розглянемо кілька прикладів генерування випадкових чисел за допомогою методів лінійної конгруентності.

Приклад 1:

$M = 2^N$
$k = 3 + 8 \cdot q$ (или $k = 5 + 8 \cdot q$)
$b = 0$
r_0 — непарно

Було виявлено, що кілька псевдовипадкових чисел, згенерованих з даних прикладу 1, повторюватимуться кожні M / 4 числа. Перед обчисленням число q задається довільно, але слід пам’ятати, що ряд створює враження випадкового при великому k (тобто q). Якщо b непарний і k = 1 + 4 q, результати можуть дещо покращитися – у цьому випадку ряд повторюватиметься через кожні M чисел. Після довгих пошуків k зупинився на 69069 та 71365.

Приклад 2 :

$M = 2^{31} - 1$

k = 1 220 703 125

b = 7

$r_0 = 7$

Генератор випадкових чисел , що використовує дані прикладу 2 , виведе окреме випадкове число з періодом 7 мільйонів.

1.3 Тести роботи генераторів

Від якості роботи ГВЧ залежить якість роботи всіх систем і точність результатів. Отже, випадкові послідовності, створені за допомогою ГВЧ, повинні відповідати ряду критеріїв. Існує два види перевірок:

- перевірити рівномірність розподілу;

- Статистичні перевірки незалежності.

Перевірити рівномірність розподілу. ГВЧ має давати значення статистичних параметрів, близькі до наступних характеристик рівномірного випадкового закону:

$m_r = \frac{\sum_{i=1}^{n} r_i}{n} \approx 0.5$ — математичне очікування;

$$D_r = \frac{\sum_{i=1}^{n} (r_i - m_r)^2}{n} \approx 0.0833$$ — дисперсія;

$$\sigma_r = \sqrt{D_r} \approx 0.2887$$ — середньоквадратичне відхилення

Частотний тест. Частотний тест дозволяє дізнатися, скільки чисел потрапляє в інтервал (mr - σr; mr + σr), тобто (0,5 - 0,2887; 0,5 + 0,2887) або нарешті (0,2113; 0,7887). Оскільки 0,7887 - 0,2113 = 0,5774, ми робимо висновок, що при хорошому ВЧ у цьому інтервалі приблизно 57,7% усіх падаючих випадкових чисел має бути випущено (див. рис 1.4).

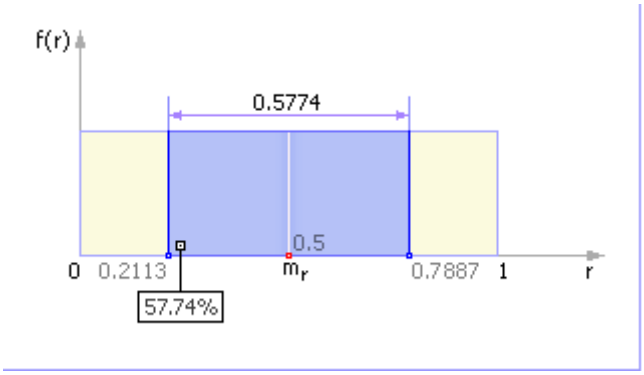

Рис 1.4 - Частотна діаграма ідеального ГВЧ у разі перевірки його на частотний тест

Статичний тест на незалежність. Розглянемо приклад. Випадкове число 0,3223456123 складається з числа 3223456123, а число 0,4423123543 складається з числа 4423123543. У поєднанні з послідовністю чисел маємо: 322345612344231235543.. Зрозуміло, що теоретична вірогідність рі випадання і- ой цифри (від 0 до 9) рівна 0.1

Далі підраховують частоту зустрічальності кожної цифри у вилученій експериментальній послідовності. Наприклад, число 1 падає 2 рази з 20, а число 6 падає 5 разів з 20.

Перевірка появи серій із однакових цифр. Нехай nL позначає кількість послідовних рядів чисел довжини L. Необхідно перевірити всі L від 1 до m, де m — це число, задане користувачем: число того самого числа, яке найчастіше зустрічається в ряді. У прикладі «32234561234423123543» знайдено 2 ряди довжиною 2 (22 і 44), тобто n2 = 2 і 2 ряди довжиною 3 (444 і 333), тобто n3 = 2. Ймовірність ряду довжин в L дорівнює: pL = 9 · 10 - L (теоретична). Тобто ймовірність довжини послідовності одного символу дорівнює: p1 = 0,9 (теоретичне значення). Ймовірність двох символьних послідовностей дорівнює: p2 = 0,09 (теоретична). Ймовірність того, що послідовність із трьох символів є довгою, дорівнює: p3 = 0,009 (теоретична). Наприклад, ймовірність довжини послідовності одного символу дорівнює pL = 0,9, оскільки з 10 може бути тільки один символ і лише 9 символів (нулі не враховуються). Імовірність того, що два однакових символу «XX» зустрінуться послідовно, дорівнює 0,1·0,1·9, тобто вірогідність 0.1 того, що в першій позиції з'явиться символ "X", множиться на вірогідність 0.1 того, що в другій позиції з'явиться такий же символ "X" і множиться на кількість таких комбінацій.

1.4 Інші методи

Розглянемо послідовність

$$xn = (a_1x_{n-1}+\ldots+a_kx_{n-k}) \pmod 2. \quad (1.2)$$

Характеристичним поліномом цієї послідовності є $P(z) = z^k – a_1z^{k-1} - \ldots - a_k$. Це лінійна кругова залежність у полі Z_2, що складається з двох елементів (0 і 1). Це кругове відношення називається регістром зсуву і має період довжини $p = 2^k -1$ тоді і тільки тоді, коли P - це примітивний поліном. Генератор на зсувному регістрі називається генератором з вихідною послідовністю.

$$U_n = \sum_{i=1}^{L} x_{ns+i-1}\, 2^{-i} \quad (1.3)$$

де розмір шагу s та длина слова L – цілі позитивні числа.

Це говорить про те, що генератори на основі регістра зсуву швидкі та мають тривалі періоди, якщо фундаментальний тричлен таких генераторів вибрано правильно. Тому вони особливо підходять для додатків, де потрібна велика кількість випадкових чисел. Проте в таких генераторах були виявлені кореляції, що може призвести до систематичних помилок у розрахунках Монте-Карло.

Метод Фібоначчі з запізнюванням.

Цей метод дозволяє отримати більш високу “якість” псевдовипадкових чисел. Причина , чому датчики Фібоначчі є найбільш популярними , полягає в тому , що швидкість реальних арифметичних операцій порівнюється зі швидкістю цілочисельних операцій , а датчики Фібоначчі природно реалізовані в реальній арифметиці . Існують різні сценарії відкладення використання методу Фібоначчі . Один з найпоширеніших датчиків Фібоначчі заснований на такій круговій формулі:

$$k_i = \begin{Bmatrix} k_{i-a} - k_{i-b}, \text{якщо } k_{i-a} \geq k_{i-b} \\ k_{i-a} - k_{i-b} + 1, \text{якщо } k_{i-a} < k_{i-b} \end{Bmatrix} \quad (1.4)$$

де k_i – дійсне число в діапазоні [0,1], a, b – цілі додатні числа, параметри генератора. Для роботи датчика Фібоначчі необхідно знати максимальне значення a, b з попередньо згенерованих випадкових чисел. У програмній реалізації потрібен певний обсяг пам’яті для зберігання згенерованих випадкових чисел, залежно від параметрів a і b.

Приклад. Обчисліть послідовність перших десяти чисел, згенерованих методом Фібоначчі із затримкою, починаючи з k5, у наступних вихідних даних a = 4, b = 1, k_0=0.1; k_1=0.7; k_2=0.3; k_3=0.9; k_4=0.5:

$k_5 = k_1 - k_4 = 0.7 - 0.5 = 0.2;$
$k_6 = k_2 - k_5 = 0.3 - 0.2 = 0.1;$
$k_7 = k_3 - k_6 = 0.9 - 0.1 = 0.8;$
$k_8 = k_4 - k_7 + 1 = 0.5 - 0.8 + 1 = 0.7;$
$k_9 = k_5 - k_8 + 1 = 0.2 - 0.7 + 1 = 0.5;$

$k_{10} = k_6 - k_9 + 1 = 0.1 - 0.5 + 1 = 0.6;$
$k_{11} = k_7 - k_{10} = 0.8 - 0.6 = 0.2;$
$k_{12} = k_8 - k_{11} = 0.7 - 0.2 = 0.5;$
$k_{13} = k_9 - k_{12} + 1 = 0.5 - 0.5 + 1 = 1;$
$k_{14} = k_{10} - k_{13} + 1 = 0.6 - 1 + 1 = 0.6.$

Ми бачимо, що згенерована послідовність чисел виглядає як випадкова послідовність. Насправді дослідження підтверджують, що отримані випадкові числа мають хороші статистичні властивості. Для генераторів, створених за методом відкладеного Фібоначчі, рекомендовані параметри a і b, які, можна сказати, пройшли перевірку якості. Наприклад, дослідники пропонують такі значення: (a, b) = (55, 24), (17, 5) або (97,33). Якість отриманих випадкових чисел залежить від значення константи a: чим воно більше, тим вища просторова розмірність, в якій зберігається однорідність випадкового вектора, утвореного отриманими випадковими числами. При цьому зі збільшенням значення константи a збільшується і обсяг пам'яті, який використовує алгоритм. Тому для простих застосувань рекомендоване значення (a, b) = (17,5). Значення (a, b) = (55, 24) дозволяє отримати числа, які задовольняють більшості криптографічних алгоритмів, вимогливих до якості випадкових чисел. Значення (a, b) = (97,33) дозволяють отримувати дуже якісні випадкові числа і використовуються в алгоритмах, працюючих з випадковими векторами високої розмірності.

Генератори ПВЧ, грунтовані на методі Фібоначчі із запізнюванням, використовувалися для цілей криптографії. Крім того, вони застосовуються в математичних і статистичних розрахунках, а також при моделюванні випадкових процесів. Генератор ПВЧ, побудований на основі методу Фібоначчі із запізнюванням, використовувався в широко відомій системі Matlab.

Метод Парка-Міллера

Найпростіша послідовність, яку можна запропонувати для реалізації генератора рівномірного розподілу :

$$I(j+1)= a * I(j) \pmod m \quad (1.5)$$

з відповідним постійним відбором. Парк і Міллер запропонували постійну

$$a=7^5=16807,\ m=2^{31}-1=2147483647$$

Цей підхід можна застосувати безпосередньо на мові асемблера, але мови високого рівня можуть виявити переповнення. Щоб уникнути цього, Шарге пропонує часткове розкладання модулів. Модулі розбиті на вирази:

$$m = a * q+ r$$

Якщо $r<q$ і $0<z<m - 1$, то при цьому величини $a* (z \bmod q)$ і $r*[z/q]$ завжди лежать в інтервалі 0,...,m - 1. Для множення $(a*z) \pmod m$ при цьому використовується алгоритм:

$$t = a (z \bmod q) - r[z/q]$$

якщо $t<0$, то $t += m$.

$$(a*z)(\bmod m)=t. \quad (1.6)$$

Вихор Мерсенна.

Генератор псевдовипадкових чисел (PPS), розроблений японськими вченими Макото Мацумото (яп. Matsumoto Zhen) і Нішимура Такудзі (яп. Nishimura Takuji) у 1997 році. Вихори Мерсенна засновані на властивості простих чисел Мерсенна (звідси назва) і швидко генерують високоякісні псевдовипадкові числа за допомогою випадковості. Завихрення Мерсенна не мають багатьох недоліків, властивих іншим ГПВЧ, таких як короткий час циклу, передбачуваність і легко виявлені статистичні закономірності. Однак цей генератор не зашифрований, що обмежує його використання в криптографії. Існує принаймні два загальних варіанти цього алгоритму, які відрізняються лише значенням використаних простих чисел Мерсенна, найпоширенішим з яких є алгоритм MT19937, який має період $2^{19937}-1$ (близько $4,3 * 10^{6001}$).

Вихор Мерсенна являє собою скручений узагальнений регістр зворотного зв'язку (TGFSR) . «Вихор» — це перетворення, яке забезпечує рівномірний розподіл псевдовипадкових чисел, згенерованих у 623 вимірах (для лінійних конгруентних генераторів воно обмежене 5 вимірами). Таким чином, кореляційна функція між двома зразками послідовностей у вихідній вихровій послідовності Мерсенна є незначною. Псевдовипадкові послідовності, створені вихорами Мерсенна, мають тривалий період, рівний числу Мерсенна, що більш ніж достатньо для багатьох практичних застосувань. Ефективна реалізація Вихора Мерсенна перевищує швидкість багатьох стандартних ГПВЧ (зокрема, вона в 2-3 рази швидше лінійних конгруентних генераторів). Алгоритм вихрових струмів Мерсенна використовується в програмних бібліотеках Boost , Glib і C++, Python , Ruby , R , PHP , MATLAB , Autoit .III

Алгоритм ISAAC

ISAAC (Indirection, Shift, Accumulate, Add and Count) — це алгоритм псевдовипадкових чисел, принципи якого важче запам’ятати, ніж принципи IA та IBAA, але він має багато переваг перед ними . Коли ISAAC був розроблений, йому був представлений наступний список вимог:

- стабільність пароля;
- неможливо отримати внутрішній стан наявних вихідних результатів;
- відсутність коротких періодів;
- немає тенденції в розподілі бітів протягом циклу;

Упорядковані стани повинні швидко стати хаотичними. На відміну від більшості генераторів псевдовипадкових чисел на основі потокового шифру, ISAAC не призначений для використання регістрів лінійного зсуву зі зворотним зв’язком. Середня кількість машинних інструкцій, необхідних для отримання 32-розрядного значення, становить 18,75. 64-розрядна версія

ISAAC (ISAAC-64) вимагає 19 інструкцій для отримання 64-розрядного значення.

ISAAC має масив S , який визначає внутрішній стан системи , також випадковим чином розташований у масиві 2^n елементів від 0 до $2^n - 1$ довжини K біт , ітератор і три змінні a, b і c відповідають попередньому стану генератора, а вихідні дані z і S є масивами однакової довжини. Однак, крім цих змінних, змінні які вводяться p0,p1,p2,p3 , які визначають значення функції , що залежить від двох ітераторів:

$$f(a,i) = \begin{cases} a << p_0, & \text{if } i \equiv 0 \bmod 4 \\ a >> p_1, & \text{if } i \equiv 1 \bmod 4 \\ a << p_2, & \text{if } i \equiv 2 \bmod 4 \\ a >> p_3, & \text{if } i \equiv 3 \bmod 4 \end{cases}$$

Алгоритм BBS

Алгоритм Блум - Блюм - Шуб (BBS) - це генератор псевдовипадкових чисел, запропонований Ленор Блум, Мануель Блум і Майкл Шуб у 1986 році.

BBS має наступний вигляд :

$$x_{n+1} = x_n^2 \bmod M,$$

де - M = pq — добуток двох великих простих чисел p і q. На кожному кроці алгоритму вихідні дані отримуються з x_n шляхом взяття біта парності або одного чи більше молодших бітів x_n.

Два простих числа p і q повинні бути порівняні з 3 за модулем 4 (це гарантує, що кожне квадратне віднімання має квадратний корінь, який також є квадратним відніманням) і найбільший спільний дільник НОД φ (p-1), φ (q-1) має бути малим (це збільшує довжину циклу). Цікавою особливістю цього алгоритму є те, що якщо ви знаєте початковий стан генератора x0 і числа p і q, ви можете отримати xn без обчислення всіх n - 1

попередніх чисел. n-е значення можна обчислити «безпосередньо» за такою формулою:

$$x_n = x_0^{2^n \bmod ((p-1)(q-1))} \bmod M.$$

Цей генератор хороший для криптографії, але не для моделювання, оскільки він недостатньо швидкий. Однак він має виключно високу стабільність, що забезпечується якістю генератора на основі обчислювальної складності задачі чисельного розкладання. Якщо прості числа вибираються ретельно, а вихідні дані O(loglogM) біт на xn, то межа, яку приймає M, швидко зростає, і обчислення вихідних бітів буде таким же складним, як і розкладання M на множники. Якщо цілочисельне розкладання однаково складно (як очікувалося), то BBS з великим M не створить жодних невипадкових шаблонів, які можна виявити за допомогою достатньої кількості обчислень. Однак може виникнути швидкий алгоритм факторізації, тому немає гарантії, що BBS є надійним.

2 АНАЛІЗ ПАРАМЕТРІВ ЛІНІЙНО КОНГРУЄТНОГО МЕТОДУ

2.1 Вибір параметрів

Вибрані параметри можна розділити на 3 типи: відмінний, хороший і поганий. Розглянемо кожен тип:

- Відмінно. Це параметри, які повністю задовольняють умовам теореми. Усі бітові тести пройшли успішно, і загальний розподіл бітів на бін був близьким до 0,5. Максимальний цикл досягнуто, потужність >= 5. Усі різні числа в послідовності з'являються рівно один раз. Все вищесказане говорить про хороший генератор. Приклад - a = 6645; c = 2543; m = 65536

-В порядку. Це параметри, які частково задовольняють умовам теореми. Усі бітові тести пройшли успішно, і загальний розподіл бітів був близьким до 0,4 x 0,6 або 0,5 у більшості або в кожному бункері. Деякі з цих параметрів можна визначити як різні, але вони різні – не може бути максимального періоду та частоти зустрічей (число може бути більше 1, а може не з'являтися взагалі). Приклад - a = 91; c = 91; m = 65536 (поточний період 32768)

- Погано Аргумент цього типу не задовольняє умовам теореми, і тест не виконується. Є короткий період. Загальна кількість 0 і 1 у двійкових цифрах вищого порядку сильно варіюється. Послідовність не виглядає випадковою або виродженою візуально. Приклад - a = 28; c = 2; m = 45.

Вибір модуля.

Період не може перевищувати m. Тому m має бути достатньо великим. Переважно m дорівнює найбільшому елементу в послідовності imax + 1 (плюс один, оскільки підрахунок і починається з нуля). Ще одним популярним вибором є метрика 2n. Припускаючи, що n є довжиною в бітах машинного слова, операцію отримання залишку можна опустити. Якщо m є мірою 2, операцію залишку можна замінити швидшою побітовою операцією

І (хоча це не має значення для сучасних комп’ютерів): x% 2n == x & (2n - 1)
Приклад (x - ціле число):

- x% 2 == x & 1;
- x% 4 == x & 3;
- x% 8 == x & 7.

Часто для m вибирають одне з простих чисел Мерсенна. При виконанні обчислень з 32-розрядними даними зазвичай використовується число 2^{31} - 1.

Вибір множника a.

Коефіцієнт слід вибирати так, щоб період був якомога довшим. Цей фактор має серйозну проблему: при малих значеннях a, якщо поточний елемент послідовності досить малий, ймовірно, що наступний елемент також буде малим.

Вибір інкремента c.

Цей варіант можна вибрати довільно. Багато разів він встановлюється на нуль, але тривалість періоду зменшується і X0! = 0

Початкове значення (seed)

Отже, ми знаходимо, що в послідовності є цикл. У нашому прикладі період дорівнює лише чотирьом елементам. В інших випадках цикл може бути довгим, але це завжди! Тобто при генерації випадкових чисел за допомогою методу лінійної конгруентності, якщо нам потрібна дуже велика послідовність, значення будуть повторюватися з певною періодичністю. У нас є ефективний генератор випадкових чисел. Ми використовуємо його для різних програм. Завжди, завжди цей генератор створює ту саму послідовність чисел (при однакових початкових значеннях). Ми можемо встановити лише початкові значення.

Зрозумілою мовою .Вибрано 4 номери:

- модуль m (m > 0);
- множник a (0 <= a <m);
- приріст c (0 <= c <m);
- Початкове значення X0 (0 <= X0 <m)

Послідовність заснована на використанні такої кругової формули: $X_n + 1 = (a * X_n + c) \bmod m$. Цей метод дає дуже хороші псевдовипадкові числа, але якщо ми візьмемо числа m, a, c довільно, результат, швидше за все, нас розчарує. Коли m = 7, X0 = 1, a = 2, c = 4, ви отримаєте таку послідовність: 1,6,2,1,6,2,1,. Очевидно, ця послідовність не зовсім відповідає визначенню випадковості. Однак ця невдача приводить нас до двох важливих висновків:

- Метод лінійної конгруентності дає нам повторювані послідовності. Насправді, будь-яка функція, яка представляє скінченну множину X в X, дасть значення, яке повторюється в циклі;
- Числа m, a, c, X0 не повинні бути випадковими

Потім, наше завдання — максимізувати унікальну частину послідовності (До речі, очевидно, що довжина єдиної частини не може бути більшою за m).

Не вдаючись у подробиці доказів, припустимо, що період послідовності буде рівним m лише за умови виконання наступних трьох умов:

- Числа c і m відносно прості;

- Для кожного простого p, яке є дільником m, a дорівнює 1 разу на p;

- Якщо m кратне 4, то a - 1 має бути кратним 4.

Коротше кажучи, про метод лінійної конгруенції слід сказати, що отримана за його допомогою послідовність, хоча й випадкова в достатньому сенсі, не є криптографічно стабільною. Оскільки, знаючи 4 послідовних числа, криптоаналітик може побудувати систему рівнянь, з яких можна знайти a, c, m.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ

3.1 Реалізація лінійно конгруєтного методу

За основу програмної реалізації усіх методів була взята мова програмування - C++ . Робоче середовище – Microsoft Visual Studio 2022.

У цій програмі описані усі кроки , які потрібно зробити для отримання випадкових чисел , а саме :

- Вибір множника , інкремента , початкового значення ;
- Генерація нового необробленого випадкового числа ;
- Генерація наступного випадкового числа ;
- Отримання послідовності випадкових чисел.

Уся генерація та обробка випадкових чисел робиться у наступному методі congruential :

```
cout << "Час роботи для обчислення даного об'єму(N) = " <<
                search_time / 1000.0 << " секунд" << endl;
        system("pause");
        return 0;
}
long double congruential(unsigned long& x)
// функція генерації чисел
{
        //cout << x << "\n"; // перевірка корректності числа
        x = ((a * x) + c) % m;
        // формула линійного конгруэнтного методу генерації випадкових
        // чисел
        return (x / long double(m));
```

Результат праці алгоритму ви можете побачити далі (див . рис 3.1)

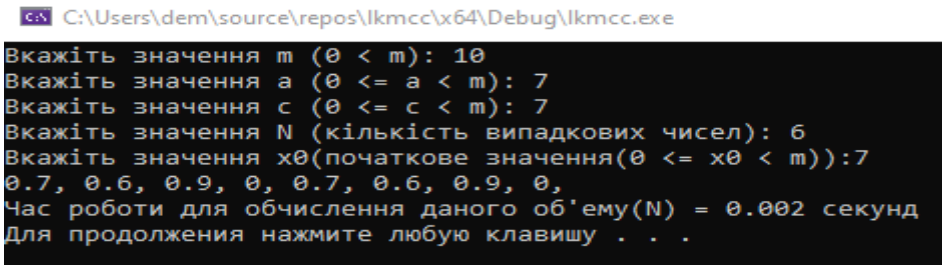

Рис 3.1 – Результат роботи лінійно конгруєтного методу

Увесь програмний код буде розташований у додатку А.

3.2 Реалізація лінійно-інверсного методу

Реалізація цього методу полягає у обчисленні послідовності випадкових чисел X_n у кільці класів за модулем натурального числа n.

Основна відмінність від лінійного методу полягає в тому, що він використовується для генерування послідовності чисел, яка є оберненою до попереднього елемента, а не до попереднього.

Параметрами генератора є :

$seed$ — сіль
a — множник $(0 \leqslant a < n)$
b — приріст $(0 \leqslant b < n)$

У випадку простого n

$$x_0 = seed$$
$$x_{i+1} \equiv (ax_i^{-1} + b) \mod n \quad \text{if } x_i \neq 0$$
$$x_{i+1} = b \quad \text{if } x_i = 0$$

У випадку складного n

$$x_0 = seed$$
$$x_{i+1} \equiv (ax_i^{-1} + b) \mod n$$

Параметри підбираються таким чином , щоб

$$(seed, n) = 1; (a, n) = 1.$$

Основний метод роботи лінійно-інверсивного методу:

```
long double congruential(unsigned long& y)
// функція генерації випадкових чисел
{
        unsigned long d = (unsigned long)pow(y, 2),
```

```
        mod = (unsigned long)pow(p, m);
    long double dmod = pow(p, m);
    //cout << y << " "; // перевірка корректності
    cout << "(";
    cout << (y / dmod) << ", ";
    y = ((a / y) + b + (c * d)) % mod;
    return (y / dmod);
}
```

Результат праці алгоритму ви можете побачити далі (див рис 3.2)

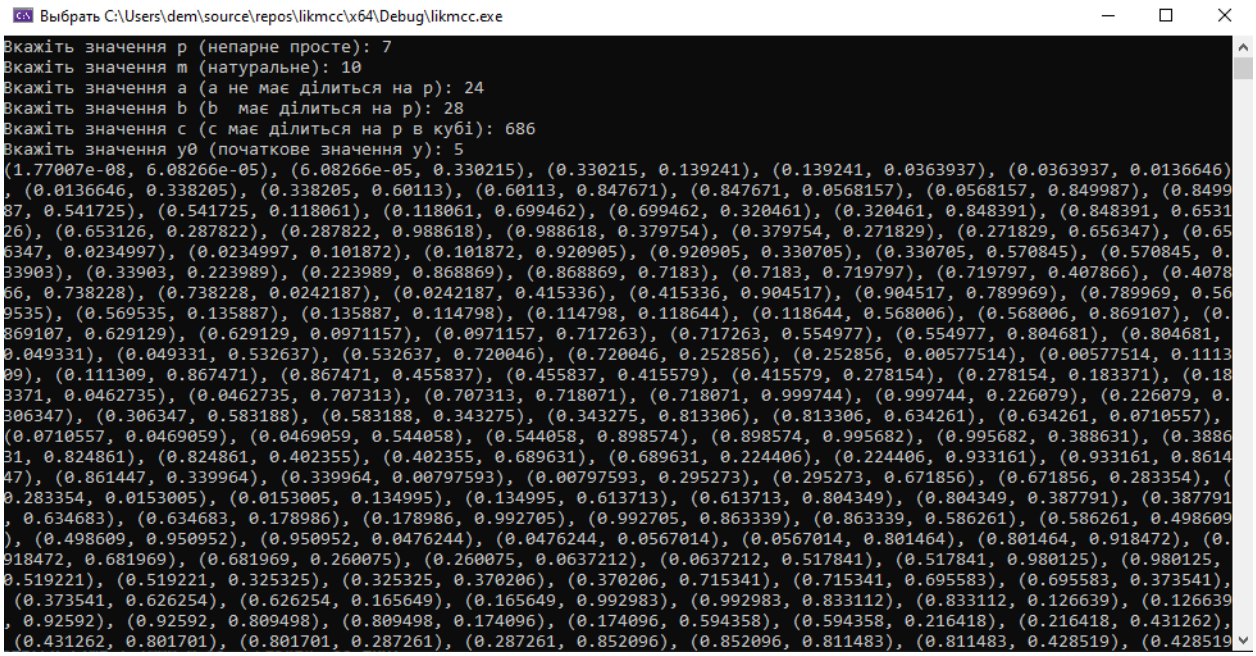

Рис 3.2 – Результат лінійного-інверсивного метода.

Увесь програмний код буде розташований у додатку Б.

3.3 Реалізація методу Фібоначчі с запізнюванням

Лаги a і b не слід вибирати довільно. Рекомендуються такі значення відставання: (a, b) = (55,24) , (17,5) або (97,33) . Якість отриманих випадкових чисел залежить від значення константи, і чим більша константа, тим вища просторова розмірність, в якій зберігається однорідність випадкового вектора, утвореного отриманими випадковими числами. При цьому зі збільшенням значення константи a збільшується і обсяг пам’яті, який використовує алгоритм. Для простих додатків, які не використовують високорозмірні вектори з випадковими компонентами, рекомендується

значення (a,b)=(17,5) . Значення (a, b) = (55,24) дозволяє отримати числа, які задовольняють більшість алгоритмів, які вимагають якості випадкових чисел. Значення (a, b) = (97,33) дозволяє отримати дуже якісні випадкові числа для випадкових векторів великої розмірності. Описаний фибоначчиїв генератор випадкових чисел (з лагами 20 і 5) використовується в широко відомій системі Matlab.

Основний метод роботи :

```
srand(time(NULL));
	int* Arr = new int[max(lagA, lagB)];
	for (int i = 0; i < max(lagA, lagB); i++)
		Arr[i] = rand();
	for (int i = Count; i > 0; i--)
	{
		int Res; //Результат
		if (Arr[max(lagA, lagB) - lagA] >= Arr[max(lagA, lagB) - lagB])
		{
			Res = Arr[max(lagA, lagB) - lagA] - Arr[max(lagA, lagB) -
lagB];
		}
		else Res = Arr[max(lagA, lagB) - lagB] - Arr[max(lagA, lagB) -
lagA];

		for (int i = 0; i < max(lagA, lagB); i++)
			Arr[i] = rand();

		cout << Res << endl;

	}
```

Результат роботи ви можете подивитися далі (див рис 3.3)

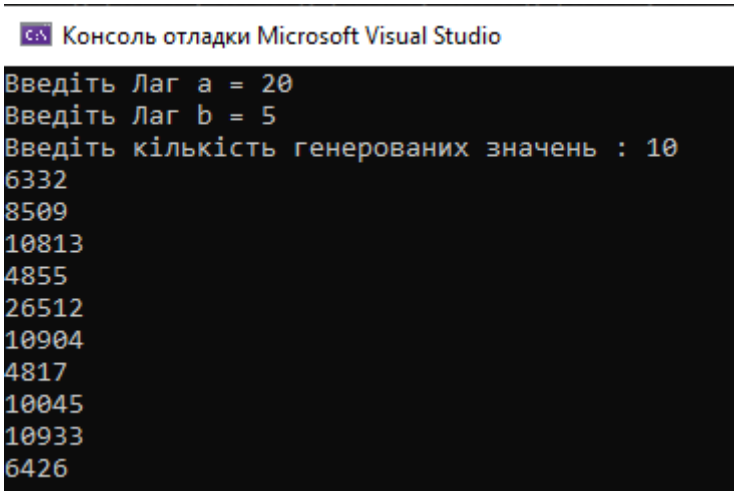

Рис 3.3 – Результат роботи методу Фібоначчі

Увесь програмний код буде розташований у Додатку В.

ВИСНОВОК

В результаті дипломної роботи є програмне забезпечення Реалізовано алгоритм генератора псевдовипадкових чисел.

Реалізуються такі методи: Метод лінійної конгруенції, Метод Фібоначчі Затримка та віднімання (адитивний генератор), інверсний послідовний підхід.

Програма виконує всі необхідні функції і дозволяє Автоматизація процесу генерування псевдовипадкових чисел. Методи генерування псевдовипадкових чисел та досліджено їх випробування. Досліджено параметри методу лінійної конгруенції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кнут Д. Искусство программирования, том 2. Получисленные алгоритмы. Вильямс, Вашингтон, США, 2001.

2. Lehmer D. Proceedings of a Second Symposium on Large Scale Digital Calculating Machinery. Harvard University Press, Cambridge, Massachusetts, USA, 1951, 141-146.

3. Gauss C. Disquisitiones Arithmeticae. Yale University Press, New Haven, Connecticut, USA, 1965, §90-92.

4. Green B., Smith K., Klem L. Empirical Tests of an Additive Random Number Generator. Journal of the Association for Computing Machinery, Vol. 6, New York, USA, 1959, 527-537.

5. Eichenauer J., Lehn J. A non-linear congruential pseudo random number generator. Statistische Hefte, Springer Berlin Heidelberg, Vol. 27, Berlin, Germany, 1986, 315-326.

6. Luscher M. A portable high-quality random number generator for lattice field theory simulations. Computer Physics Communications, Vol. 79, Elsevier Science, Amsterdam, Netherlands, 1994, 100-110.

7. Todd J., Taussky O. Generation and testing of pseudo-random numbers. Symposium on Monte Carlo methods, University of Florida, Wiley, New York, USA, 1956, 15-28.

8. Marsaglia G. A current view of random number generators. Computer science and statistics: Symposium on the Interface 16, Atlanta, Georgia, USA, March 1984, Elsevier Science, Amsterdam, Netherlands, 3-10.

9. Niederreiter H. Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992.

ДОДАТОК А. КОД ПРОГРАММИ РЕАЛІЗАЦІЇ ЛІНІЙНОГО КОНГРУЕТНОГО МЕТОДА

```
#include "stdafx.h"
#include <iostream>
#include <clocale>
#include <ctime>
using namespace std;
long double congruential(unsigned long&);

unsigned int m,
a,
c;
int N;
int main(int argc, char* argv[])
{
	setlocale(LC_ALL, "Ukrainian");
	cout << "Вкажіть значення m (0 < m): ";
	cin >> m;
	while (m <= 0)
	{
		cout << "m має бути більше нуля";
		cout << "\n";
		cout << "Вкажіть значення m (0 < m): ";
		cin >> m;
	}
	cout << "Вкажіть значення a (0 <= a < m): ";
	cin >> a;
	while (a > m || a < 0)
	{
		cout << "Введене a більше m або меньше нуля ";
		cout << "\n";
		cout << "Вкажіть значення a (0 <= a < m): ";
			cin >> a;
	}
	cout << "Вкажіть значення c (0 <= c < m): ";
	cin >> c;
	while (c > m || c < 0)
	{
		cout << "Введене c більше m або менше нуля";
		cout << "\n";
		cout << "Вкажіть значення c (0 <= c < m): ";
		cin >> c;
	}
```

```
	cout << "Вкажіть значення N (кількість випадкових чисел): ";
	cin >> N;
	while (N < 0)
	{
		cout << "Введене N меньше нуля";
		cout << "\n";
		cout << "Вкажіть значення N (кількість випадкових чисел): ";
		cin >> N;
	}
	unsigned long x0;
	cout << "Вкажіть значення x0(початкове значення(0 <= x0 < m)):";
	cin >> x0;
	while (x0 > unsigned int(m) || x0 < 0)
	{
		cout << "Введене x0 більше m або меньше нуля";
		cout << "\n";
		cout << "Вкажіть значення x0(початкове значення(0 <= x0 < m)):";
		cin >> x0;
	}
	unsigned int start_time = clock();
	cout << (x0 / long double(m)) << ", ";
	for (int i = 0; i <= N; i++)
		cout << congruential(x0) << ", ";
		cout << "\n";
	unsigned int end_time = clock();
	unsigned int search_time = end_time - start_time;
	cout << "Час роботи для обчислення даного об'єму(N) = " <<
		search_time / 1000.0 << " секунд" << endl;
	system("pause");
	return 0;
}
long double congruential(unsigned long& x)
// функція генерації чисел
{
	//cout << x << "\n"; // перевірка корректності числа
	x = ((a * x) + c) % m;
	// формула линійного конгруэнтного методу генерації випадкових
	// чисел
	return (x / long double(m));
	}
```

ДОДАТОК Б . КОД ПРОГРАМИ РЕАЛІЗАЦІЇ ЛІНІЙНО ІНВЕРСНОГО ГЕНЕРАТОРА

```
#include "stdafx.h"
#include <iostream>
#include <cmath>
#include <clocale>
#include <ctime>
using namespace std;
long double congruential(unsigned long&);
// прототип функции генерации случайных чисел
int isprostoe(int);
// прототип функции опередеделения простое ли число
int p,
m,
a,
b,
c;
int main(int argc, char* argv[])
{
	setlocale(LC_ALL, "Russian");
	cout << "Вкажіть значення p (непарне просте): ";
	cin >> p;
	while (isprostoe(p) == 0)
	{
		cout << "Число не просте";
		cout << "\n";
			cout << "Вкажіть значення p (непарне просте): ";
		cin >> p;
	}
	cout << "Вкажіть значення m (натуральне): ";
	cin >> m;
	while (m < 1)
	{
		cout << "Число не натуральне";
		cout << "\n";
		cout << "Вкажіть значення m (натуральне): ";
		cin >> m;
	}
	cout << "Вкажіть значення a (a не має ділиться на p): ";
	cin >> a;
	while (a % p == 0)
	{
		cout << "Число a ділиться на p";
```

```
            cout << "\n";
            cout << "Вкажіть значення a (a не має ділиться на p): ";
            cin >> a;
        }
        cout << "Вкажіть значення b (b  має ділиться на p): ";
        cin >> b;
        while (b % p != 0)
        {
            cout << "Число b не ділиться на p";
            cout << "\n";
            cout << "Вкажіть значення b (b має ділиться на p): ";
            cin >> b;
        }
        cout << "Вкажіть значення c (c має ділиться на p в кубі): ";
        cin >> c;
        while (c % (int)pow(p, 3) != 0)
        {
            cout << "Число c не ділиться на p в кубі";
            cout << "\n";
            cout << "Вкажіть значення c (c має ділиться на p в кубі): ";
            cin >> c;
        }
        const int N = 2 * (int)pow(3, 8);
        // кількість випадкових чисел (13122)
        unsigned long y0;
        cout << "Вкажіть значення y0 (початкове значення y): ";
        cin >> y0;
        while (y0 > (unsigned int)pow(p, m) || y0 < 0)
        {
            cout << "введене y0 більше p^m або меньше нуля";
            cout << "\n";
            cout << "Вкажіть значення y0(початкове значення(0<=y0<p^m)):
";
            cin >> y0;
        }
        unsigned int start_time = clock();
        for (int i = 0; i <= N; i++)
            cout << congruential(y0) << "), ";
        cout << "\n";
        unsigned int end_time = clock();
        unsigned int search_time = end_time - start_time;
        // искомое время
        cout << "Час роботи для цього об'єму(N) = " <<
            search_time / 1000.0 << " секунд" << endl;
```

```
        system("pause");
        return 0;
}
long double congruential(unsigned long& y)
// функція генерації випадкових чисел
{
                unsigned long d = (unsigned long)pow(y, 2),
                mod = (unsigned long)pow(p, m);
        long double dmod = pow(p, m);
        //cout << y << " "; // перевірка корректності
        cout << "(";
        cout << (y / dmod) << ", ";
        y = ((a / y) + b + (c * d)) % mod;
        return (y / dmod);
}
int isprostoe(int pp) // вертає 1, якщо p-просте и 0,
// якщо p не просте
{
        int d;
        if (pp < 2) return 0;
        if (pp == 2) return 1; // 2 - просте
        if (pp % 2 == 0) return 0; // парне -> не просте
        d = 3; // початковий дільник
        while (d * d <= pp)
        {
                if (pp % d == 0) return 0;
                d += 2; // наступний дільник
        }
        return 1; // дільников не знайдено -> просте
                                                }
```

ДОДАТОК В . КОД РЕАЛІЗАЦІЇ МЕТОДА ФІБОНАЧЧІ С ЗАПІЗНЮВАННЯМ.

```
#include <iostream>
#include <time.h>

using namespace std;

unsigned int lagA = 0, lagB = 0;

int main(void)
{
	int Count = 0;
	setlocale(LC_ALL, "Ukrainian");
	cout << "Введіть Лаг a = ";
	cin >> lagA;
	cout << "Введіть Лаг b = ";
	cin >> lagB;
	cout << "Введіть кількість генерованих значень : ";
	cin >> Count;


	srand(time(NULL));
	int* Arr = new int[max(lagA, lagB)];
	for (int i = 0; i < max(lagA, lagB); i++)
		Arr[i] = rand();
	for (int i = Count; i > 0; i--)
	{
		int Res; //Результат
		if (Arr[max(lagA, lagB) - lagA] >= Arr[max(lagA, lagB) - lagB])
		{
			Res = Arr[max(lagA, lagB) - lagA] - Arr[max(lagA, lagB) -
lagB];
		}
		else Res = Arr[max(lagA, lagB) - lagB] - Arr[max(lagA, lagB) -
lagA];

		for (int i = 0; i < max(lagA, lagB); i++)
			Arr[i] = rand();

		cout << Res << endl;

	}
```