НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ» ІМЕНІ ІГОРЯ СІКОРСЬКОГО
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

„Розробка програм аналізу та генерації псевдовипадкових чисел“

ua

Курсова робота
дисципліна
„Алгоритмізація та програмування“

Виконавець:
Федурко Микола КА-75
Керівник:
Романов В. В.

“Захист дозволено”
“_____”_____ 2018 р.
Захищено з оцінкою:

Київ 2018

Завдання

Розробити програму генерації псевдовипадкових чисел в залежності від вибраної функції розподілення з можливістю роботи з файлами (запис та зчитування функції), а також результатів генерації.

Зміст

Вступ

В еру розквіту комп'ютерних технологій, цей світ стає все більш визначеним. Скільки пройшло часу з зародження кібернетики як всесвітньо відомої науки, а знаходження незакономірних чисел до цих пір дуже потрібне в цій і багатьох інших галузях.

Задача знаходження випадкових чисел виникла в найнеочікуваніших областях кібернетики і прикладного програмного забезпечення. Починаючи з криптографії і закінчуючи використанням в лотереях.

Якщо вдуматися в рішення цієї задачі, то можна легко прийти до очевидного рішення - використовувати фізику для генерації випадкових значень. Можна використовувати шуми навколишнього середовища, або вологість повітря в тихому океані, як пропонують нам відомі ресурси генерації випадкових чисел.

Це рішення даної проблеми, не зовсім коректне, адже шуми і скачки вологості - цілком детерміновані(визначені) фізичні явища. З теоретичної точки зору знаючи початкові умови замкнутої системи можна визначити всі її подальші тани в будь-який момент часу. Для цього лише потрібна достатня потужність ЕОМ(Електронно-обчислювальної машини). Мусимо зауважити, що існує відносно молода сім'я теорій квантової фізики, яка стверджує, що явища мікросвіту(світу квантів) можна описувати лише за допомогою теорії ймовірності, тобто випадково. Але оскільки людство поки що не може впевнитися в цьому, ми опустимо цей варіант.

Також можливі ситуації, коли використання фізики для вирішення задачі неможливе, або не вигідне. Мається на увазі обмеженість в ресурсах вбудовуваних систем. Адже, підключення додаткових пристроїв і завантаження коду в «польових» умовах не завжди можливе.

Отже приходимо до задачі створення програмних продуктів для генерації випадкових значень засобами математики. Що на перший погляд є не такою вже й складною задачею. Візьмемо наприклад число яке за спостереженнями цілковито випадкове та нескінченне. Тут же приходимо до висновку, що щоб взяти випадкову цифру потрібно згенерувати випадковий номер цієї цифри, заходимо в замкнуте коло.

Саме тому логічним кроком буде створення продукту для такої генерації засобами математичного апарату, щоб знаючи всі попередні значення згенерованих даних не мати можливості прогнозувати наступне, що і є метою даної роботи.

Для проекту була обрана мова програмування «C», саме через швидкість своєї роботи та лаконічний синтаксис.

Розділ 1

Обгрунтування і вибір алгоритмів

Прийшовши в попередньому розділі до необхідності створення програмного забезпечення, логічно описати ряд вимог, яким мусить задовольняти обраний алгоритм.

1.1 Означення

Спочатку введемо деякі означення і скорочення:
ГПВЧ - генератор псевдовипадкових чисел, програмне забзепечення, що задовольняє умові непередбачуваності, детальніше описаній в останньому абзаці вступу. Відрізняється від ГВЧ(генератора випадкових чисел) цілковитою визначеністю.

Оскільки ми використовуємо математичний апарат, то цілковитої невизначенності не вийде досягнути. Дійсно, коли ми генеруємо послідовність, то кожен наступний член визначається попереднім, отже, чисто теоретично можлива ситуація коли послідовність почне зациклюватись з певним періодом. Далі буде показано, що це неминуча ситуація.

Період ГПВЧ - кількість елементів між першим входженням даного елемента і його другим входженням.

Також потрібно все ж таки якимось чином задати початкове значення, адже в протилежному випадку всі значення будуть однакові.

Ключ ГПВЧ - початкове значення послідовності випадкових чисел. Також відомий як seed (англ. - зерно).

1.2 Вимоги

З уж е чітко окресленою термінологією доцільно оголосити список вимог, яким мусить задовольняти наш алгоритм ГПВЧ:

- Достатньо довгий період, що гарантує відсутність зациклювання в межах цієї задачі. Довжина періоду мусить бути математично-доведеною.
- Портабельність - дана реалізація мусить функціонувати на різних системах однаково.
- Відтворюваність - можливість відтворити дану послідовність довільну кількість разів. Це особливо важливо при проведенні різних статистичних дослідів, для можливості записати результати.
- Ефективність - швидкий час роботи алгоритму, та малі затрати за пам'яттю
- Швидкість отримання X_{n+i} елемента послідовності чисел за відомими Xnелементом для довільного i. Це дозволяє розділяти послідовність на декілька потоків(послідовностей).
- Також існують декілька тестів для перевірки послідовності на «випадковість», до таких відносяться:
- DIEHARD
- NIST
- Тести Дональда Кнута

- Тест на останній біт - один із тестів для перевірки ГПВЧ на криптостійкість(властивість алгоритму протистояти криптоаналізу). Сам тест звучить так: якщо не існує поліноміального(зі складністю $O(n^k)$ алгоритму, який знаючи перші k біт послідовності зможе знайти $k+1$ біт з ймовірністю більше 50%, то такий ГПВЧ не пройшов тесту.

Попри велику множину графічних, статистичних тестів, кількість яких все росте, в 1982 році китайський вчений Ендрю Ян довів [2], що ГПВЧ, який пройшов «тест на останній біт» пройде також і всі інші тести, виконувані за поліноміальний час.

Зациклення будь-якої послідовності ПВЧ прийняте, як постулат в підрозділі (1.1) мусить бути доведене.
Доведення:

ГПВЧ, можна розглянути, як функцію f на множині X (формула 1.1)

$$f : X \mapsto X \tag{1.1}$$

З елементарних комбінаторних міркувань це відображення має скінченну кількість комбінацій. Тобто якою б великою множиною не було X, є на порядки більша (але, що важливо скінченна) кількість кроків, після яких буде відбуватись повтор. Що і необхідно було довести.

1.3 Розгляд алгоритмів генерації псевдовипадкових чисел

Після постановки ряду вимог до наших алгоритмів, логічним продовженням буде розгляд декількох алгоритмів, та відбір з них тих, що найкраще підходять під вище описані потреби.

1.3.1 Лінійний конгруентний метод

Одним із простих і популярних методів на сьогодні є лінійний конгруентний метод (ЛКМ), запропонований Д. Г. Лехмером в 1949 році.

В його основі лежить вибір чотирьох ключових чисел:

- $m > 0$, - модуль;
- $0 \leq a \leq m$, - множник;
- $0 \leq c \leq m$, - приріст (інкремент);
- $0 \leq X_0 \leq m$, - початкове значення(seed);

Послідовність ПВЧ, яка отримується формулою (1.2) називається лінійною конгруентною послідовністю (ЛКП). Ключом(seed) для неї слугує X_0.

$$X_{n+1} = (a \cdot X_n + c) \, mod \, m, \; n \geq 1 \tag{1.2}$$

В даній формулі надзвичайно важливий мудрий підбір параметрів. Наприклад при $X_0 = 7, a = 8, c = 9, m = 10$ отримуємо послідовність 7, 5, 9, 7, 5, 9, 1, ... Послідовність взагалі не виглядає випадковою. Переконливі і цікаві ілюстрації, до цих прикладів можна знайти у [1].

Існує узагальнення формули ЛКМ:

$$X_{n+1} = (a^k \cdot X_n + \frac{a^k - 1}{a - 1} \cdot c) \, mod \, m, \; k \geq 0, \; n \geq 0 \tag{1.3}$$

Доцільно розглянути задачу правильного підбору параметрів.Не вдаючись в доведення, яке було проведено в [1] та [3], можна стверджувати:

- мусить бути достатньо великим
- значення m розумно вибирати рівним $2q$, де q - число бітів в машинному слові, оскільки це дозволяє не приміняти ділення по модулю в формулі (1.3)

Теорема (2.1.) Лінійна конгруентна послідовність, визначена параметрами m, a, c, X_0, має період m т. т. т. к.(тоді і тільки тоді, коли): Числа і m взаємно прості;

1. $(a - 1) \vdots p$, для деякого простого p, яке є дільником m

2. $(a-1) \vdots 4$, якщо $m \vdots 4$

Доведення даної теореми, було освітлено в [3]. Також, дізнатися про список «вдалих» констант для цього методу та цікаві факти з його історії, можна в [4]. Підводячи підсумки:

Переваги:

- Простота реалізації
- Універсальність

Швидкість Недоліки:

- Простота «взлому» показана в [5] та [6]

Тим не менше ЛКМ, виявився доволі корисним, для не криптографічних задач, таких як моделювання та ігрові програми. Не дивлячись на його проблему, на сьогодні його використовують в таких мовах програмування, як Java, C, C++, C#.

1.3.2 Реєстр зсуву з лінійним зворотнім зв'язком. Метод М-послідовності

Наступний клас ГПВЧ містить в основі ідею перетворення бітів деякого числа. Реєстр зсуву - впорядкований набір бітів, допускаючий операцію зміни позиції цих бітів на одну й ту ж величину.

Реєстр зсуву з лінійним зворотнім зв'язком (РЗЛЗЗ) - реєстр зсуву бітових слів, у якого вхідний біт, є лінійною функцією інших бітів. Вхідний біт стає в комірку з номером нуль. Кількість комірок p називають довжиною реєстру.

Для натурального числа p і a1,a2,... ,ap-1 , що приймають значення 0 або 1, визначена рекурентна формула (1.4)

$$X_{n+p} = a_{p-1}X_{n+p-1} + a_{p-2}X_{n+p-2} + \cdots + a_1X_{n+1} + X_n \qquad (1.4)$$

Як видно з формули (**??**), для РЗЛЗЗ функція зворотнього зв'язку є лінійною булевою функцією, залежною від стану всіх чи декотрих бітів реєстру.

Одна ітерація алгоритму, який генерує послідовність, складається з наступних кроків:

1. Зміст комірки p - 1 формує формує черговий біт ПСП бітів.
2. Зміст комірки 0 визначається значенням функції, обчислюваним за формулою (1.4).
3. Зміст кожного i-го біта переміщується в $(i+1)$-й, $0 \leq i < p-1$.
4. В комірку 0 записується значення з кроку 2.

Найбільше додатнє значення N таке, що $X_{n+N} = X_n \forall n$називають періодом послідовності. Цю послідовність називають M-послідовністю, якщо її період рівний $(2p-1)$. Буква «M» в назві від англійського «maximum». Дійсно, адже максимальний можливий період і є періодом цієї послідовності. На рис. 1.1 зображено алгоритм роботи даного ГПВЧ для формули (1.4), де $F(x)$ — лінійна функція зворотнього зв'язку:

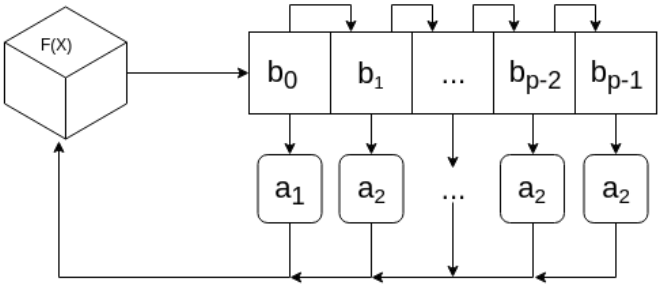


Рис. 1.1: Схема роботи РЗЛЗЗ

Визначимо, які вхідні дані відповідають M-послідовності(з найбільшим періодом), для цього введемо означення:

Многочлен $F(x)$ степені p називається примітивним, якщо він не ділить націло многочлен виду $(x^s - 1)$, де $s < 2^p - 1$.

Найменше натуральне число e при якому $x^e - 1 \vdots F(x)$ називається показником многочлена.

Властивості послідовності бітів залежать від многочлена (1.4)

- Якщо старший коефіцієнт $a_p = 0$, то періодичність згенерованої послідовності може проявитися не одразу.

- Якщо старший коефіцієнт $a_p = 1$, то послідовність буде періодичною з самого початку
- Якщо $k = 1$, то примітивність многочлена (1.4) є необхідною і достатньою умовою того, що ПСП згенерована РЗЛЗЗ буде мати максимальну довжину $N = 2^{p-1}$.

Підсумовуючи:
Переваги:

- Швидкість
- «Хороші» статистичні властивості
- Простота реалізації на програмному рівні

Недоліки:

- Повільна програмна реалізація
- Можливість взлому за [7] та [8]
- РЗЛЗЗ використовуються у вбудовуваних системах, саме через простоту апаратної реалізації.

1.3.3 Вихор Мерсенна

Метод Вихор Мерсенна був запропонований в 1997 році японськими вченими Макото Мацумото та Такудзі Нісімура. Введемо деякі означення: «Вихор» - перетворення, що забезпечує рівномірний розподіл ПВЧ. Число Мерсенна - натуральне число Mn, визначене формулою (1.5):

$$M_n = 2^n - 1 \tag{1.5}$$

Однією із важливих властивостей цих чисел є те, що якщо Mn просте, то n - також просте. Є багато варіацій цього алгоритму, розглянемо MT19937 - найпопулярніший варіант. По суті, даний ГПВЧ

є РЗЛЗЗ, який складається з 624 комірок по 32 біта, та генерується за формулою (1.6):

$$X_{n+p} = X_{n+q} \bigoplus (X_n^r | X_{n+1}^l) A \ \ (n = 0, 1, 2, ...) \tag{1.6}$$

Де

- p, q, r — цілі константи, p — степінь рекурентності, $1 \leq q \leq p$;
- X_n — w-бітне двійкове число
- $(X_n^r | X_{n+1}^l)$ - двійкове ціле число, отримане конкатенацією(об'єднанням) чисел
- X_n^r та X_{n+1}^l, де перші $(w - r)$ бітів отримані із X_n, а останні r бітів із X_{n+1}
- A - матриця нулів і одиниць порядка w, визначена за допомогою формули (1.7).
- XA - добуток, при обчисленні якого спочатку виконують операцію $X \gg 1$(зсув на один вправо), якщо останній біт дорівнює 0, потім коли останній біт дорівнює 1, обчислюється $XA = (X \gg 1)a$.

$$\begin{aligned} a &= (a_{w-1}, a_{w-2}, \cdots, a_0) \\ X &= (x_{w-1}, x_{w-2}, \cdots, x_0) \end{aligned} \tag{1.7}$$

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & & & 0 \\ \vdots & \cdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & & \ddots & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & & a_0 \end{pmatrix}$$

Кроки виконання алгоритму Мерсенна:

1. Ініціалізація значень u, h, a за формулами (1.8)

$$\begin{aligned} &u = (1, 0, \cdots, 0) \text{— всього } (w - r) \text{ біт.} \\ &h = (0, 1, \cdots, 1) \text{— всього } r \text{ біт.} \\ &a = (a_{w-1}, a_{w-2}, \cdots, a_0) \text{ — остання строка матриці } A \end{aligned} \tag{1.8}$$

2. Ініціалізація $X_0, X_1, \cdots, X_{p-1}$

3. $$Y = (y_0, y_1, \cdots, y_{w-1}) \tag{1.9}$$

,

4. Обчислення нового значення X_i:

$$\begin{aligned} X_n &= X_{(n+q)\ mod\ p} \bigoplus (Y \gg 1) \bigoplus a, \text{ якщо найменший біт } y_0 = 1; \\ X_n &= X_{(n+q)\ mod\ p} \bigoplus (Y \gg 1) \bigoplus 0, \text{ якщо найменший біт } y_0 = 0; \end{aligned} \tag{1.10}$$

5. Обчислення $X_j T$:

$$\begin{aligned} Y &= X_n; \\ Y &= Y \bigoplus (Y \gg u); \\ Y &= Y \bigoplus ((Y \ll s) \cdot b); \\ Y &= Y \bigoplus ((Y \ll t) \cdot c); \\ Z &= Y \bigoplus (Y \ll l). \end{aligned} \tag{1.11}$$

Z подається на вихід як результат.

6. $n = (n + 1)\ mod\ p$, переходимо на крок 3 Параметри алгоритму, ще з початку підібрані авторами, щоб давати найкращі результати. w обирається за розміром машинного слова, тому для

64-бітної версії формула має інший вигляд. Параметри методу Вихор Мерсенна:

$$p = 624,\ w = 32,\ r = 31,\ q = 397,$$
$$a = 2567483615(9908B0DF16),\ u = 11,$$
$$s = 7,\ t = 15,\ l = 18\ \ b = 2636928640(9D2C568016),$$
$$c = 4022730752(EFC6000016)$$

.

Підсумовуючи:
Переваги:

- Вже відкорельовані параметри, для «хорошого» результату
- Великий період, а саме $(2^{19937} - 1)$
- Хороший статистичний розподіл
- Швидший за інші алгоритми

Недоліки:

- Складність аналізу
- Складність реалізації

Мусимо зауважити, що цей же алгоритм використовується в ГПВЧ мови програмування PHP. Також є багато його варіацій з нахилом на різні параметри, такі як TinyMT(пам'ять), CryptMT(криптостійкість).

1.4 Вибір алгоритмів та конкретизація ПЗ

Більшість інших видів генераторів можна переглянути в додатку 1. В основному це різноманітні комбінації двох вище описаних методів(РЗЛЗЗ і ЛК). Оскільки РЗЛЗЗ і лінійні конгруенті генератори формують деякий «базис» ГПВЧ, то доцільно розглянути саме їх, як приклад в цій роботі. Оскільки РЗЛЗЗ існує також багато видів, то зупинимося на найбільш практичному - Вихрові Мерсенна. А лінійний конгруентний генератор оберемо звичайним, оскільки це буде яскравим простим прикладом свого класу генераторів.

Потрібно буде зробити функції для генерації чисел в обох ГПВЧ. Для ЛК зробимо функцію, яка буде т естувати параметри A, C, M формули (1.2). Також для лінійного конгруентного метода розробимо функцію, що буде «ламати» генератор, тобто отримувати множину всіх можливих значень A, C, M. Можна було б розробити такі ж

функції і для MT, але цей генератор уже збалансований на великий період і всі його змінні заздалегідь відомі, тобто все що треба зробити для його «зламу» - це реверсувати дії «загартовування», а саме крок 6 в пункті 1.3.3

Також доцільно буде написати деякі допоміжні функції. Вони описані в наступному розділі(пункт 2.1), разом зі структурами та алгоритмами для рішення поставлених задач.

Розділ 2

Розробка програми

Після надання теоретичних відомостей та обрання алгоритмів, логічно приступити до розробки алгоритмів та створення продукту.

2.1 Опис допоміжних функцій

Оскільки функції, які ми обрали в пункті 1.4 потребують «допоміжних» функцій, то доцільно спочатку описати останні, адже при опису процесу розробки основних функцій, ми будемо посилатися на цей розділ. Чесно кажучи процес розробки проходить навпаки, спочатку пишеться код, а потім, за необхідності, додаються функції, щоб структурувати цей код. Отже, в цьому пункті ми дамо алгоритми роботи цих функцій, але їхнє призначення буде розкрито в наступних пунктах.

2.1.1 Генератор фізично-випадкових чисел

int getAbsStartRand();

Функція яка повертає випадкові числа. Для цього, якщо програма на unix-подібних системах, то вона повертає значення папки «/dev/urandom», яка чиатється, як прилад та містить «шуми» комп'ютера у вигляді байтів. Якщо ж використовується Windows, то викликається схожа функція під Windows. Таким чином створюються абсолютно випадкові значення. Якщо ж відповідна функція

повернула код помилки, то ми просто повертаємо *clock()* - час від початку роботи програми, який уже є менш випадковим значенням.

2.1.2 Інші функції

Дамо перелік «простих» функцій, що також використовуються в більш складніших:

- НСД - найбільший спільний дільник
- НСК - найбільше спільне кратне
- MOD - повертає завжди додатнє значення остачі від ділення числа на число
- Перевірка на простоту числа
- Знаходження максимального числа серед чотирьох чисел

Всі оголошення описаних вище функцій помістимо в заголовковий файл *RandomFunctions.h*. А їх код помістимо в файл *RandomFunctions.c*. Див. додаток 2. Для цієї частини програми нам потрібні такі заголовкові файли:

- *stdlib.h* - бібліотека стандартних функції «C»
- *math.h* - функція для роботи з математикою
- *time.h* - функція для роботи з часом (функція *clock()*)
- *inttypes.h* - додає деякі типи та специфікатори цих типів до функції *printf()*
- *sys/random.h, fcntl.h, unistd.h* - для виклику *getrandom(...)*, лише для роботи під Unix
- *Wincrypt.h* - для виклику *CryptGenRandom(...)*, лише для роботи під Windows

2.2 Реалізація методу «Вихор Мерсенна»

Частина програми, що відповідає за роботу Вихру Мерсенна складається з двох файлів: *BIZ_MT.c*, та *BIZ_MT.h*. В другому розміщені оголошення функцій, макросів, структур, а в першому самі функції.

2.2.1 Структура MTRand

Згідно з пунктом 1.3.3 створена структура MTRand, що зберігає значення констант і змінних для генерації чисел. Вона складається з таких полів:

- *int32_t *newx* — масив для нової генерації значень
- *unsigned int arr_len* — P (в позначеннях пункту 1.3.3)
- *unsigned int offset* — Q (в позначках пункту 1.3.3)
- *unsigned int hardening0* — U (в позначках пункту 1.3.3)
- *unsigned int hardening1* — S (в позначках пункту 1.3.3)
- *int32_t hardening2* — B (в позначках пункту 1.3.3)
- *unsigned int hardening3* — T (в позначках пункту 1.3.3)
- *int32_t hardening4* — C (в позначках пункту 1.3.3)
- *unsigned int hardening5* — L (в позначках пункту 1.3.3)
- *int32_t * x* — масив початкових значень для генерації
- *int32_t lower_mask* — h (в позначках пункту 1.3.3)
- *int32_t higher_mask* — u (в позначках пункту 1.3.3)
- *int32_t matrix_row* — A (в позначках пункту 1.3.3)

Існує глобальний екземпляр цієї структури *MTRand mt_def*, над яким і будуть відбуватись зміни, що дуже схоже на «машину станів», яка використовується наприклад в OpenGL.

2.2.2 Функції ініціалізації використання та видалення генератора

MTUserArray * mtRandInit(MTRand * rand);

Функція яка ініціалізує структуру *mt_def* початковими значеннями. Їй передається вказівник на структуру *rand*, якщо він дорівнює нулю, то *mt_def* — ініціалізується значеннями «за замовчуванням»(описані вище константи). Потім виділяється пам'ять для масиву *mt_def->newX* і *mt_def->X*. Другий заповнюється початковими випадковими значеннями за допомогою функції *getAbsStartRand()* описаної в пункті 2.1.1.

Якщо ж користувач передає вказівник на свою структуру *MTRand * rand*, то всі значення ініціалізуються користувацькими даними. Якщо користувач передав свою структуру, то логічно подумати, що у нього є свої початкові значення, тобто пам'ять виділяється лише під масив *rand->newX*, при чому вказівник *mt_def->newX* вказує на ту ж область пам'яті, для того щоб спокійно працювати з нею і надалі. Адже всі взаємодії здійснюються за допомогою *mt_def*.

Далі створюється вказівник на структуру *MTUserArray * MTArr*, під який виділяється пам'ять та поля якого заповнюються відповідно:

- *MTArr->ptr = mt_def.newX;*
- *MTArr->len = mt_def.arr_len;*

Ця структура повертається, це зроблено для зручної роботи користувача, адже таким чином він має доступ до згенерованих значень. Мусимо зауважити, що в функцію передається саме вказівник, щоб надати можливість обрати режим «за замовчуванням».

void mtDefRand();

Оскільки генерація відбувається крок в крок за алгоритмом описаним в пункті 1.3.3, то немає ані найменшого сенсу повторюватися. Проте звертаємо увагу, на те, що нове значення записується шляхом математичних дій над *mt_def->x* та записується в *mt_def->newX*. То-

му, щоб повторити цикл, користувачу необхідно викликати функцію *void mtDefSwapBuffers();*

void mtDefKill(MTUserArray * MTArr);

На момент видалення у користувача є структура із згенерованими значеннями, пам'ять для якої була виділена динамічно, тобто її потрібно вивільнити(принаймні в «C»). Він передає вказівник, а ми очищаємо пам'ять під нього, також вивільняючи пам'ять масивів *mt_def->x*, *mt_def->newX*.

2.3 Реалізація лінійного конгруентного генератора

Частина програми, що відповідає за роботу лінійного конгруентного генератора складається з двох файлів: *BIZ_LKM.c*, та *BIZ_LKM.h*. В другому розміщені оголошення функцій, макросів, структур, а в першому самі функції.

2.3.1 Структури мови «C» використані при реалізації

struct LKMSet;

Має три поля: a, c, m. Використовується для зберігання пар A, C, M формули (1.2). Надалі для більшості змінних цієї частини програми використовуються тип «int64_t» заголовкового файлу «inttypes.h». В заголовковому файлі уже є 15 варіантів «хорошого» вибору параметрів, тобто їх можна використовувати у своїх програмах.

struct LKMUserArray;

void free_array(LKMUserArray ** arr);

Має два поля: *len, set. set* - вказівник(в перспективі - массив) структур *LKMSet. len* — довжина цього масиву. Використовується для взаємодії користувача з функціями «зламу» генератора. Оскільки в основному в програмі будуть використані вказівники на цю структуру, то єдина незначна функція цього пункту відповідає за очищення

динамічно виділеної пам'яті під цю структуру.

struct LKMRand;

Має два поля: *seed, set. set* — структура описана в першому абзаці. *seed* - початкове значення для генератора. Тобто ця структура є користувацьким генератором, який він може протестувати, привести в дію, зламати. В заголовковому файлі «за замовчуванням» оголошений екземпляр цієї структури *lkm_ def*, який(як описано на сторінці 20) створює абстракцію машини станів.

2.3.2 Функція ініціалізації і використання генератора

void lkmRandInit(LKMrand * rand);

rand — користувацькі параметри, згруповані в структуру. Коли користувач передає вказівник на структуру, то структура *mt_ def* приймає свої значення за наступними правилами: Початкове значення(*lkm_ def.seed*) присвоюється остача від ділення користувацького початкового значення(*rand->seed*) на *rand->set.m*. Що зроблено для профілактики помилок, коли користувач задасть початковим значенням число більше *rand->set.m*. Те ж відбувається і з параметрами *lkm_ def.set.a, lkm_ def.set.c*. Значення *rand->set.m* присвоюється *lkm_ def.set.m* Якщо ж користувач передасть в функцію *NULL*, то всі параметри *lkm_ def* заповняться константами «за замовчуванням». Окрім *seed*'а який буде отриманий за допомогою нашого генератора випадкових чисел, опис якого наданий в пункті 2.1.1. Ще раз наголосимо, що в функцію передається вказівник для можливості реалізації значення «за замовчуванням»(детальніше описано в останньому абзаці пункта 2.2.2).

int64_t lkmDefRand();

Функція генерації чітко описується формулою (1.2). Нове значення присвюється *lkm_ def.seed* і повертається.

2.3.3 Опис функції аналізу обраних параметрів

void lkmTestParam(LKMSet * set);

Функція яка отримує на вхід набір значень, обробляє його і дає корисні поради для покращення результатів. Під «результатами» мається на увазі довжина періоду згенерованих значень. Для обробки вона керується принципами описаними в пункті 1.3.1. Виводить всі зауваження, якщо таких немає, то хвалить користувача за вдалий вибір. Якщо такі все ж були, то вона пропонує обрати з можливих «хороших» значень описаних в заголовковому файлі. Якщо ж користувач і цього не хоче, то вона пропонує ввести нові значення самостійно. В разі обрання нових значень вона запускає себе ще раз через *goto START*. В протилежному випадку функція закінчує свою дію.

Є неабияке бажання наголосити, що оператор переходу «goto» використовується для лаконічності і зрозумілості коду, оскільки:

- Використовувати ще один зовнішній цикл і змінні для керуванням користувацьким рішенням дуже погано впливає на читабельність коду
- Використання рекурсії призведе до неясності у виводі програми, оскільки після завершення всіх функцій, вони будуть декілька раз виводити результати своєї роботи, та і взагалі рекурсія - ресурсозатратна річ, оскільки вимагає заміну стека.

START - дуже зрозуміле ім'я для переходу

2.3.4 Опис функції «зламу» генератора

void lkmCrack(LKMUserArray ** arr, int64_t x0, int64_t x1, int64_t x2, int64_t x3);
Функція приймає на вхід користувацький масив для запису в нього списку можливих параметрів. Та чотирьох чисел, уявлення про які краще отримати із системи (2.1)

$$\begin{cases} X_1 &= (A * X_0 + C)\ mod\ M; \\ X_2 &= (A * X_1 + C)\ mod\ M; \\ X_3 &= (A * X_2 + C)\ mod\ M; \end{cases} \tag{2.1}$$

У нас є три невідомі змінні A, C, M, які будуть записані в *(*arr)->set[i]*. Де i - деяке ціле число, яке залежить від кількості можливих варіантів цього генератора.

Алгоритм роботи цієї функції чітко описаний в [9], тому немає сенсу описувати його тут, також його опис може бути легко зрозумілий дивлячись ' на коментарі до коду в додатку 2.

2.3.5 Опис функції зменшення списку можливих варіантів

void lkmRedList(LKMUserArray ** arr);

Оскільки після роботи функції з попереднього пункту ми отримали масив можливих варіантів параметрів, то логічно, що вибагливий користувач може бути не задоволений кількістю варіантів, які треба перебрати. Хоча риптоаналітики будуть раді і цій інформації. Саме тому було прийняте рішення написати ще одну функцію, яка буде зменшувати (з англ. reduce) кількість можливих рішень «зламу» генератора.

Суть її полягає в тому, що вона запрошує дані для «тестування»(формула (1.2)) і перевіряє кожну трійку параметрів зі списку на відповідність цим даним. Під даними мається на увазі X-и послідовності згенерованих чисел.

Якщо вони не відповідають, то значення m відповідного номеру зі списку встановлюється в -1. Оскільки множина роботи функції з формули (1.2) — натуральні числа, то -1 — чудовий виборі для попередження про «невідповідність» конкретної трійки шуканому генератору.

Якщо кількість «відповідних» варіантів зменшилось до нуля, то десь була допущена помилка і програма повідомить вам про це.

Якщо залишився лише один «відповідний» варіант, то програма привітає вас з успішним «зламом» вашого генератора і завершить роботу.

Розділ 3

Керівництво користувачу

При користуванні функціями створеного ПЗ, потрібно керуватись наступними правилами:

1. Вводити значення тих типів, яких в тому чи іншому випадку потребує програма.

2. Надавати програмі реальних значень змінних, оскільки при наданні некоректних хідних даних програма, в більшості випадків, буде повертати неправильні вихідні дані

3. Краще прислухатися до порад аналізатора вхідних даних

Оскільки даний програмний продукт створений, як основа для розробки подальших програм, то керівництво користувачу логічно обмежується цими правилами. Проте написання користувацьких програм полягає на плечі розробника, який в свою чергу мусить написати «Керівництво користувачу» своєї програми.

Розділ 4

Керівництво розробнику

Оскільки програма складається з декількох модулів, то їх необхідно підключити, а саме підключити такі заголовкові файли:

- RandomFunctions.h
- BIZ_MT.h
- BIZ_LKM.h

Також при компіляції потрібно додати їх у свій проект, якщо ви працюєте в IDE, або компілювати в gcc з такою командою:

```
gcc -g Gen/BIZ\_LKM/BIZ\_LKM.c Gen/BIZ\_MT/BIZ\_MT.c
Gen/RandomFunctions.c your\_prog.c  -o your\_prog -lm
```

де *Gen* - папка в якій зберігаються файли, *your_prog.c* — файл коду вашої програми У *MT* і *LKM* є *makefile*, змісти яких такі:

- MT:

```
mt: BIZ\_MT.c ../RandomFunctions.c
gcc -g BIZ\_MT.c ../RandomFunctions.c -o mt -lm
```

- LKM:

```
lkm: BIZ\_LKM.c ../RandomFunctions.c
gcc -g BIZ\_LKM.c ../RandomFunctions.c -o lkm -lm
```

Наглядно файлове розміщення показане на рис. 4.1

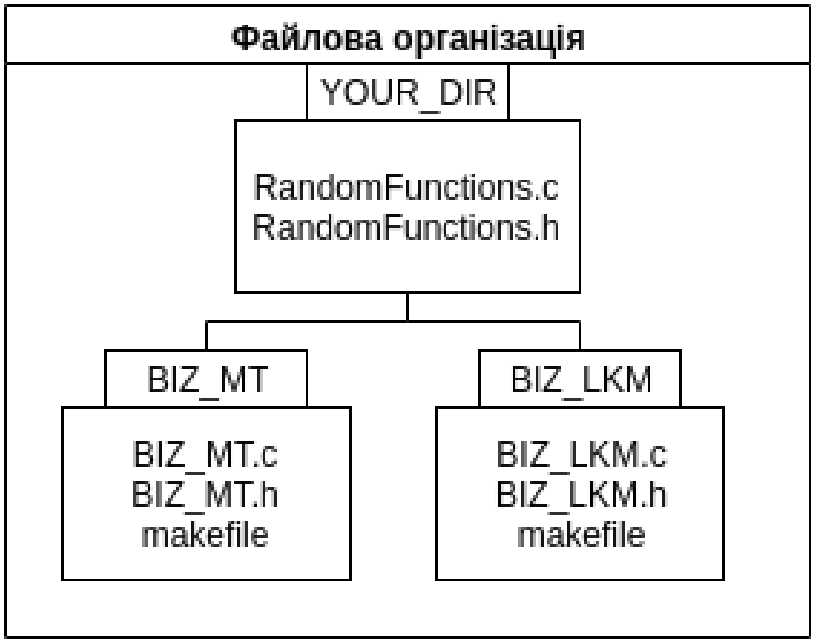


Рис. 4.1

При роботі з функціями генераторів і допоміжними функціями доцільно перечитати розділ 2, в якому все детально описано.

Розділ 5

Отримані результати

5.1 Особливості трансляції, компонування і налагодження

Програма написана на мові програмування «C» і компілювалася за допомогою *gcc(GNU Compiler Collection)* в операційній системі *Manjaro x86_64*(в основі стоїть *Arch Linux*). Налагодження відбувалося за допомогою *gdb(GNU Debugger)*. Також була використане інструментальне програмне забезпечення — valgrind, для пошуку витоків пам'яті при динамічному виділенні останньої.

Мусимо зауважити, що програма працює крос-платформенно, оскільки були використані функції із стандартної бібліотеки. І єдина платформно-залежна функція *int getAbsStartRand();* була перероблена на незалежну, шляхом розмежування її дій на кожній із систем.

5.2 Результати рахунку контрольного прикладу

Нижче представлені десять із 624 значень функції *getAbsStartRand()* поданих на вхід алгоритму *MT19937*:

247985497	580701539	1529392703	912531385	1265435855
1302975498	1287154760	743203497	1768934035	334814459

Табл. 5.1

А результати роботи цього алгоритму представлені в таблиці(5.2):

1071570338	1230189442	1995407069	1859497223	691350141
1025630334	662585518	742414615	257092565	1622530717

Табл. 5.2

Вхідне число та результати генерації 10 значень алгоритму ЛКМ представлені в таблиці (5.3):

120662	78319	303796	75293	265510
224147	47904	237981	120078	151895

Табл. 5.3

5.3 Дослідження програми на прикладах загального вигляду

Ось ще декілька прикладів і їх вхідні дані:
МТ:

194803655	1482254104	1952256794	1158780004	191592212
914618153	1580310791	355019064	1331845275	506617266

Табл. 5.4

1.

297296437	750736443	1554390564	845510788	869263591
1705663656	795895134	443239708	1015279662	1813760740

Табл. 5.5

2.

204891265	1817214921	2014320176	1511120103	720954010
54403601	1491681194	55121309	1660107234	1724766138

Табл. 5.6

3.

1252179751	2035438832	1970646593	173389379	1187127225
1292866889	19207774	74090186	1950533416	1236812472

Табл. 5.7

4.

939173219	1326844729	384281815	442109918	459537698
1067460826	1110887713	401014842	1219832291	357412613

Табл. 5.8

5.

1384074049	749476078	346328957	983151122	1417898450
264878214	245775973	1158322305	19748281	990873778

Табл. 5.9

6.

ЛКМ:

99345	227582	229439	66116	310813
143730	297067	137524	261301	101598

Табл. 5.10

1.

202071	25068	306785	21922	292179
81256	239853	188170	173407	164764

Табл. 5.11

2.

299947	194204	222281	197378	2195
283932	99289	71966	250163	289080

Табл. 5.12

3.

Висновки

В цій роботі були розглянуті декілька алгоритмів генерації псевдовипадкових чисел. Була обгрунтована сама необхідність їх використання, також стало зрозумілим чому для різноманітних галузей науки, розваг та інш. невигідно використовувати генератори фізичновипадкових чисел, та запропоноване рішення цієї проблеми.

Після розбору можна підсумувати, що ГПВЧ поділяються на два основні класи — своєрідний базис, та їх комбінації. Цими класами є:

1. РЗЛЗЗ - реєстр зсуву із лінійним зворотнім зв'язком
2. ЛКМ - лінійний конгруентний метод

Були розглянуті їх плюси та недоліки, на основі цих міркувань виникло рішення реалізувати по одному з кожного класу ГПВЧ. Так із першого ми обрали найзбалансованішу версію - MT19937. А із другого був обраний найпопулярнійший генератор - лінійний конгруентний метод.

Коли прийшла черга реалізації задуманого, то виявилося кращим рішенням - розділити кожен генератор на окремі файли, для майбутньої масштабованості проекту.

Для лінійного конгруентного методу були створені функції ініціалізації, генерації, аналізу та «зламу» генератора.

Для вихру Мерсенна були створені функції ініціалізації та генерації псевдовипадкових значень. Було прийнято рішення не «зламувати» генератор, оскільки в цій реалізації ми знаємо всі константи і це не представляє ніякого стратегічного інтересу(те ж саме з функцією аналізу).

Протягом всієї роботи була використана мова програмування «C». Оглядаючись назад можна зауважити, що для цієї роботи «зручніше» було б використовувати одну із мов у якій реалізована парадигма об'єктно-орієнтованого програмування, оскільки так код стає більше читабельним. Особливо коли мова заходить про абстракції, то функції і структури дуже зручно «обертати» в класи. Хотілось би відмітити, що фантазія породжує майже невичерпну кількість варіантів покращення коду, ось лише деякі із них:

- для пришвидшення роботи функції «зламу» генератора ЛКМ можна використати багатопотоковість(в «C» це можна зробити за допомогою POSIX threads)
- замість того, щоб використовувати структуру MTUserArray, можна звернутись за допомогою до одно(дво)зв'язних списків
- як уже було підмічено можна змінити мову на ту, що підтримує ООП, в такому разі автор може порадити «C++», оскільки вона перейняла порівняну з «C» швидкість, але набула грандіозних можливостей і все ще розвивається

Підсумовуючи: до цих пір наукою не встановлено, чи існують справжні випадкові числа, проте вони дуже потрібні і ігнорувати цю потребу не можна. Для цієї проблеми було знайдено рішення і не одне. Два основні із них були розібрані і реалізовані в даній роботі. Стало зрозумілим, що сучасні ГПВЧ — це всього лише спосіб заплутати числа, шляхом арифметичних і логічних дій над ними. Саме через це в їх назві є частка «псевдо». Хотілось би завершити висновки дуже справедливою цитатою:

> Кожен, хто використовує арифметичні методи генерування випадкових чисел, безумовно, грішить.
>
> -Джон Фон Нейман
> 1951 р.

Бібліографія

[1] Слеповичев, И. И. *Генераторы псевдослучайных чисел* / И. И. Слеповичев. – : , 2017. – 118 с.

[2] Andrew Chi-Chih Yao. *Theory and applications of trapdoor functions*. In Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982.

[3] Дональд Кнут. *Искусство программирования*, том 2. Получисленные ал- горитмы = The Art of Computer Programming, vol.2. Seminumerical Algorithms. – 3-е изд. – М.: «Вильямс», 2007. – С. 832. – ISBN 0-201-89684-2.

[4] https://goo.gl/hx3o1f

[5] J.A. Reeds, *«Solution of Challenge Cipher»*, Cryptologia, v.3, n.2, Apr 1979, pp. 83-95.

[6] .J.B. Plumstead, *«Inferring a Sequence generated by a Linear Congruence»*, Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science, 1982, pp. 153-159.

[7] https://goo.gl/EQcVW4

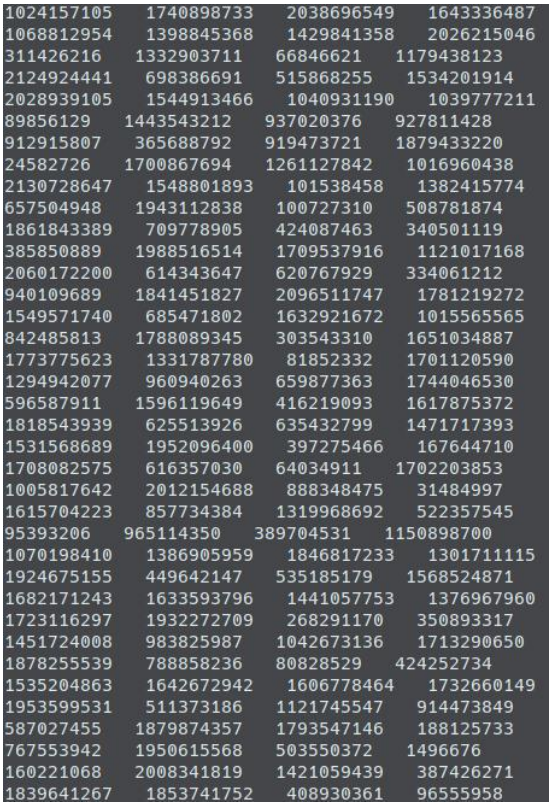
[8] https://goo.gl/hNiXt1

[9] https://goo.gl/aSTcHX

Додатки

Скріншоти роботи програми

Скріншоти програми для МТ показані на рис. 5.1(a), 5.1(b), 5.2(a), 5.2(b), 5.3, 5.4:



```
1024157105   1740898733   2038696549   1643336487
1068812954   1398845368   1429841358   2026215046
311426216    1332903711   66846621     1179438123
2124924441   698386691    515868255    1534201914
2028939105   1544913466   1040931190   1039777211
89856129     1443543212   937020376    927811428
912915807    365688792    919473721    1879433220
24582726     1700867694   1261127842   1016960438
2130728647   1548801893   101538458    1382415774
657504948    1943112838   100727310    508781874
1861843389   709778905    424087463    340501119
385850889    1988516514   1709537916   1121017168
2060172200   614343647    620767929    334061212
940109689    1841451827   2096511747   1781219272
1549571740   685471802    1632921672   1015565565
842485813    1788089345   303543310    1651034887
1773775623   1331787780   81852332     1701120590
1294942077   960940263    659877363    1744046530
596587911    1596119649   416219093    1617875372
1818543939   625513926    635432799    1471717393
1531568689   1952096400   397275466    167644710
1708082575   616357030    64034911     1702203853
1005817642   2012154688   888348475    31484997
1615704223   857734384    1319968692   522357545
95393206     965114350    389704531    1150898700
1070198410   1386905959   1846817233   1301711115
1924675155   449642147    535185179    1568524871
1682171243   1633593796   1441057753   1376967960
1723116297   1932272709   268291170    350893317
1451724008   983825987    1042673136   1713290650
1878255539   788858236    80828529     424252734
1535204863   1642672942   1606778464   1732660149
1953599531   511373186    1121745547   914473849
587027455    1879874357   1793547146   188125733
767553942    1950615568   503550372    1496676
160221068    2008341819   1421059439   387426271
1839641267   1853741752   408930361    96555958
```

(а)

```
1627426456   1734891024   387097259    1004993743
1767340100   2094777346   1327827834   1558950736
895867758    1554514697   235145168    884675449
976228941    827191812    539484371    1453380992
1936225090   606285598    389124223    1260843509
1172162990   1332823636   1591242245   1129497060
1556301239   1285479919   1596864831   899274999
1863220897   264150669    811461293    1071135182
2033567771   1820523730   1603473719   1569713685
553185518    1579321803   159203553    702077649
87345595     1462058208   1432128896   241582717
1039131309   1036799409   1850574088   1179054811
1509733146   1911894656   1250565916   773217986
292119127    164896838    300831872    492331746
1301646496   1211223640   333579935    697301932
236681395    1322846685   736876397    296645628
2000220702   1102211675   1545841012   45999377
43953355     1336974186   1782864046   1286545909
23783467     1279388027   1027920485   1499554037
1747919793   1431606472   861085510    1174624220
993290120    1976476598   1811999738   1413636755
284609698    1132087250   915639414    1448508819
1039587762   1099292245   1112885181   15585575
1890725075   1633985395   297876118    1791686415
503087398    1932994602   248634120    1179687333
1306764323   1854486446   1448044135   790259495
303132479    1313129088   122069290    133517367
99784966     1585822439   1050891702   1322157059
1315319669   1362200160   1720269563   1692480933
1384581079   2108544762   449866431    1884514551
33564953     1849480903   1639499407   1736105423
1483515769   1870249895   996631800    182029985
306972606    1321103304   2024561677   252754399
198517240    1681251231   1524456615   2138467860
1942660459   1949143600   1893146974   108605972
1434244296   1354312542   1036445450   218728863
900370872    1519558258   745196532    929308092
```

(б)

Рис. 5.1

```
428767960   794603274   841675078   1811743123
1766476722   2074612558   931001170   1917478417
1208623272   1199659065   525172281   981837319
1199018988   494607984   2128695167   292084568
1410447488   6849802   1642101973   1959210416
1587560940   358132447   1331073812   145963488
809962077   1497510198   672044443   1218104461
1183205623   559322204   840919205   279670153
1529960175   1636731963   295315686   61356158
989604437   2007478599   121578045   2029813551
1014718359   768896849   1386750581   1947795897
1152949463   1199932311   766537648   313158996
756727613   1647784020   1413672142   760620104
1610441430   1055682757   642531353   395907213
197554494   1818272600   1410973843   2099861527
402944500   153074869   1603720053   1595153575
1204617553   1324305639   1702387066   795577634
1390368047   463972457   1533160743   395981214
2133006513   260345   1073799796   393454664
2127190402   1582942202   1688210576   1985057113
1614816126   1476003345   708102838   993606471
79535004   335496467   1403010914   850133846
91928426   559675864   2051750905   1716395166
1981316350   1681345580   22518631   1643795211
1164530929   471916506   871534282   1083496289
978083745   1228201414   830324635   240858095
409810034   1074414483   1057559926   2050299731
440988523   71260222   691483602   1031457326
1040972068   53062373   639663918   1550849933
1886258558   1653120751   2075243494   1308637215
528732956   1290581247   1645606393   583804615
1500408292   386444309   1302937950   1485553250
1946338211   1050789500   2099983506   1514110565
1274139787   1765842994   64071538   841197651
94221380   1094421374   1200340157   486588768
726330597   1440846016   227195424   1926322593
1383643779   1752649518   850892956   150321318
```

(а)

```
844697309   491806273   555147636   6148297
437062080   1578548732   194895147   1100723749
718636263   1044536816   1666916908   419506810
950651299   1383586199   1040153347   611559403
1806565275   2077174535   628736569   1517549843
190206386   1704631414   1775903   1097996163
134257154   1950164557   811694228   1829525747
1943479394   1606356158   1889315802   1043979662
1871380011   679496512   153839477   2002222805
2144043798   322773666   1157260135   1917304890
173243997   1996243229   1456959694   1991364984
1959114951   1431003519   687561390   348415685
902019753   793585973   1327281995   1087867691
1535473449   903547516   344636838   1146272115
524350737   571713233   1842322957   184925138
2089279950   2097721124   297967787   452521072
1756911470   2030801413   2022685467   1738738029
652076709   1923398556   678792131   1258838807
1371181182   555773061   1371350471   2126443120
623848201   1536418965   1552003860   1674938217
1968190132   990060586   2122361570   1261196845
1726438891   877104218   799437062   1483453894
1517121736   525117390   261101225   1993615195
1223641374   2133021221   1364355446   692321481
970180014   276018230   933338328   1456144480
832178825   1412018596   1729214645   1239207071
526480320   1066940916   2101755674   1467877445
450230508   427848763   534735319   289501523
1466016113   440145355   1758659739   1571045151
79844861   306343712   2033886767   1098888625
1447432122   447769447   1632786344   158451532
1824024412   1166100562   902555313   1982152632
1627426456   1734891024   387097259   1004993743
1767340100   2094777346   1327827834   1558950736
895867758   1554514697   235145168   884675449
976228941   827191812   539484371   1453380992
1936225090   606285598   389124223   1260843509
```

(б)

Рис. 5.2

```
1839641267   1853741752   408930361   96555958
1258716204   644777396   1939895654   175997166
409401924   390309278   1782136539   146841612
510186661   1774471124   177274122   1768813096
1952603776   1514249893   497534701   657623332
745625240   1269976030   1769080397   1858626916
1992086338   949311493   1141554754   1986026012
1404783141   634942260   256922393   1469674025
1412782718   29435757   882964784   1618553
127022207   689178603   1314933250   393842781
221924332   1336322650   904187648   413117770
406860666   697355270   261832611   1578239609
2102727300   467716910   54504923   447775048
1160021553   800445864   2062970365   277826444
```

Рис. 5.3

ЛКМ для генерації сотні чисел показані на рис 5.4:

```
Ваша послідовність:
128589   19266   232963   37380
189717   19174   21951   93748
87265   7202   280259   37636
97533   147650   252687   249344
278821   81818   306035   13672
201429   20506   196603   24920
252657   99014   100191   279388
16805   57142   217099   105376
167673   56690   77127   205684
248561   11458   232075   25112
27269   117246   273243   5460
238597   269354   236431   165528
89345   117582   269439   193616
150813   258730   312067   302524
201301   66598   288615   222052
18609   159486   62883   266500
185037   5194   280671   289668
279885   226022   224779   152356
271953   103770   57507   14864
299541   159738   138155   16992
275449   309726   103023   158040
129477   31534   82911   251808
250925   107462   183119   144096
131093   191810   288447   67704
```

Рис. 5.4