МІНІСТЕРСТВО ОСВІТИ, НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

АСИМЕТРИЧНІ КРИПТОСИСТЕМИ ТА ПРОТОКОЛИ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №1

Підготували:
студенти 4 курсу
групи ФІ-84
Ковальчук О.М.
Коломієць А.Ю.

Київ – 2021

Побудова тестів для перевірки якості випадкових та псевдовипадкових послідовностей

Мета лабораторної роботи

Вивчення критеріїв згоди і набуття навичок у побудові та застосуванні тестів для перевірки статистичних властивостей бінарних випадкових і псевдовипадкових послідовностей, ознайомлення з поняттям М-послідовності.

Постановка задачі

1. Написати програми, які реалізують генератори псевдовипадкових бітів:

a) вбудований генератор вашої мови програмування;
b) генератор LehmerLow;
c) генератор LehmerHigh;
d) генератор L20;
e) генератор L89;
f) генератор Джиффі (Geffe);
g) генератор «Бібліотекар»;
h) генератор Вольфрама;
i) генератор Блюма-Мікалі BM;
j) генератор BM_bytes (байтова модифікація генератору Блюма-Мікалі);
k) генератор BBS;
l) генератор BBS_bytes (байтова модифікація генератору BBS).

2. Розробити програми для реалізації трьох тестів перевірки якості двійкових послідовностей: про рівноімовірність розподілу, про незалежність і про однорідність. Програми повинні враховувати можливості введення випробовуваної послідовності різної довжини із зазначених датчиків, а також із зовні заданого файлу, можливості задавати різні значення рівня значимості α (обов'язково: $\alpha = 0.01; 0.05; 0.1$, інші значення за бажанням).

У програмі повинне бути передбачене відображення на екрані комп'ютера результатів обробки послідовності, що перевіряється тестами, із зазначенням вхідних даних послідовності (її довжини, датчиків, що її генерують чи зовнішнього файлу), обраних значень α , обчислених та теоретичних значень χ^2 і висновку (у яких випадках послідовності відкидаються чи приймаються зазначеними критеріями).

3. Провести обробку трьома побудованими тестами прикладів послідовностей, згенерованих зазначеними датчиками, для значень $\alpha = 0{,}01; 0{,}05; 0{,}1$. Пам'ятайте, що для одержання статистично достовірних даних ваша послідовність повинна містити щонайменше мільйон бітів (краще декілька мільонів).

При багатократному проведенні експериментів ви повинні пам'ятати, що для одержання нових послідовностей з того чи іншого датчика треба використовувати нові стартові значення. В промислових реалізаціях для ініціалізації програмних генераторів псевдовипадкових чисел використовують апаратні датчики, що реалізують фізичні генератори. В межах даної роботи для ініціалізації можна використовувати інші програмні датчики або доступну ентропію (системні таймери, траєкторію миші, клавіатурний набір, ідентифікатори системних подій тощо).

Хід роботи

Опис труднощів, що виникали, та шляхів їх розв'язання

1) Для генераторів Вольфрама, «Бібліотекар», Блюма-Мікалі та Блюма-Мікалі-Шуба потрібно працювати із числами у великій арифметиці. А це складно та довго реалізувати на C++.
   Шлях розв'язання: підключили бібліотеку NTL
2) Піднесення до степеня у великій арифметиці потрібно обчислювати, використовуючи схему Горнера. Ця схема складно реалізується на C++, а у бібліотеці NTL взагалі відсутня.
   Шлях розв'язання: взяли необхідні функції із лабораторної роботи по СРОМу, де була реалізована схема Горнера. Також придумали реалізацію переведення числа типу big integer (із бібліотеки NTL) у десятковій системі числення в двійкову.
3) Операція циклічного зсуву, яка необхідна для реалізації генератора Вольфрама, не реалізована в C++.
   Шлях розв'язання: придумали специфічну реалізацію.
4) Складно зрозуміти, які саме межі будуть у циклах при реалізації, а також незрозуміло, яка саме кількість циклів нам необхідна.
   Шлях розв'язання: схематично зобразили необхідні об'єкти, які необхідно реалізувати у цьому тесті

Статистичні результати дослідження

	Test 1			Test 2			Test 3		
Alpha	**0.9**	**0.95**	**0.99**	**0.9**	**0.95**	**0.99**	**0.9**	**0.95**	**0.99**
Critical	**283.906**	**292.036**	**307.167**	**65486.6**	**65616.4**	**65858**	**3936.95**	**3968.44**	**4027.04**
bild_in_generator	273.873	273.873	273.873	64988.7	64988.7	64988.7	3786.01	3786.01	3786.01
lehmer_low_generator	0.016385	0.016385	0.016385	1.275e+08	1.275e+08	1.275e+08	1.58951	1.58951	1.58951
lehmer_high_generator	22.278	22.278	22.278	64201.9	64201.9	64201.9	2723.82	2723.82	2723.82
l_20_generator	213.578	213.578	307.167	60912.3	60912.3	60912.3	3902.26	3902.26	3902.26
l_89_generator	263.439	263.439	263.439	65490.5	65490.5	65490.5	3781.83	3781.83	3781.83
geffe_generator	218.012	218.012	218.012	72980.2	72980.2	72980.2	3655.47	3655.47	3655.47
library_generator	2.63215e+07	2.63215e+07	2.63215e+07	1.37899e+06	1.37899e+06	1.37899e+06	-nan	-nan	-nan
wolfram_generator	256.922	256.922	256.922	65052.6	65052.6	65052.6	3965.77	3965.77	3965.77
blume_mikali_bm_generator	289.275	289.275	289.275	65162.1	65162.1	65162.1	3715.09	3715.09	3715.09
bm_bytes_generator	263.391	263.391	263.391	64645.8	64645.8	64645.8	3840.76	3840.76	3840.76
bbs_generator	273.676	273.676	273.676	65768.9	65768.9	65768.9	3780.82	3780.82	3780.82
bbs_bytes_generator	240.863	240.863	240.863	64835.3	64835.3	64835.3	3965.45	3965.45	3965.45

Generators	Time work
bild_in_generator	0.543842 seconds
lehmer_low_generator	4.0237 seconds
lehmer_high_generator	4.21313 seconds
l_20_generator	0.54988 seconds
l_89_generator	0.555916 seconds
geffe_generator	0.784676 seconds
library_generator	32.9231 seconds
wolfram_generator	0.551321 seconds
blume_mikali_bm_generator	65.7603 seconds
bm_bytes_generator	68.9029 seconds
bbs_generator	1.05259 seconds
bbs_bytes_generator	4.81771 seconds

Тестування генератору* l_20_generator *на великій кількості даних

-statistics tests for alpha:0.1

- criterian_check_equinoimovirnost_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 11.7563

statistic critical: 283.906

- criterian_check_independence_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 34007.4

statistic critical: 65486.6

- criterian_for_checking_the_uniformity_of_a_binary_sequence:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 3900.85

statistic critical: 3936.95

-statistics tests for alpha:0.05

- criterian_check_equinoimovirnost_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 11.7563

statistic critical: 292.036

- criterian_check_independence_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 34007.4

statistic critical: 65616.4

- criterian_for_checking_the_uniformity_of_a_binary_sequence:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 3900.85

statistic critical: 3968.44

-statistics tests for alpha:0.01

- criterian_check_equinoimovirnost_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 11.7563

statistic critical: 307.167

- criterian_check_independence_symbols:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 34007.4

statistic critical: 65858

- criterian_for_checking_the_uniformity_of_a_binary_sequence:

hypothesis H_0 does not contradict experimental data:

statistic experimental: 3900.85

statistic critical: 4027.04

- time work program: 4.01732 seconds

Як ми можемо помітити, тест на рівноімовірність знаків на великій послідовності (8 млн біт) дав значно менше значення статистики, ніж той же самий тест на відносно невеликій послідовності (1 млн біт). Це пов'язано із тим, що **l_20_generator** є лінійним регістром, а вони, в свою чергу, генерують періодичну послідовність. І наявність циклу більш помітна на великих послідовностях, ніж на відносно невеликій послідовності в 1 млн біт.

Порівняння генераторів, що досліджувались, з точки зору статистичного аналізу та криптографічного застосування

bild_in_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти
lehmer_low_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності залежить від попереднього	Байти мають такий самий розподіл що і біти
lehmer_high_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти
l_20_generator	Генерує послідовність із рівноімовірним розподілом (не виконується лише для рівня значущості α=0.01)	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти
l_89_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього (не виконується лише для рівня значущості α=0.1)	Байти мають такий самий розподіл що і біти
geffe_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності залежить від попереднього	Байти мають такий самий розподіл що і біти
library_generator	Генерує послідовність із нерівноімовірним розподілом	Кожен наступний елемент послідовності залежить від попереднього	Байти та біти мають різні розподіли
wolfram_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти (не виконується лише для рівня значущості α=0.1)
bm_generator	Генерує послідовність із рівноімовірним розподілом (не виконується лише для рівня значущості α=0.1)	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти
bm_bytes_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти
bbs_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього (не виконується лише для рівня значущості α=0.1)	Байти мають такий самий розподіл що і біти
bbs_bytes_generator	Генерує послідовність із рівноімовірним розподілом	Кожен наступний елемент послідовності не залежить від попереднього	Байти мають такий самий розподіл що і біти (не виконується лише для рівня значущості α=0.1)

Висновки до роботи

Аналізуючи отримані результати тестів для 12 різних генераторів, можемо зробити певні висновки та припущення щодо причин не проходження того чи іншого тесту.

- Генератор **lehmer_low_generator** не проходить тест на незалежність знаків, бо, можливо, кожен його наступний елемент обчислюється рекурентно через попередній: $x_{n+1} = (ax_n + c)\ mod\ m$ і при цьому беруться саме молодші біти в якості вихідного значення. А молодші біти, теоретично, більше піддаються впливу попередньо вихідного значення.
- Генератор **library_generator** не проходить усі три тести оскільки:
  - Тест на рівноімовірність знаків, бо генерація відбувається за допомогою довільного тексту. А як ми знаємо деякі букви є менш вживаними, деякі більш вживаними. Із цього очевидно, що розподіл на алфавіті буде нерівноімовірним.
  - Тест на незалежність знаків, бо у мові є певні комбінації літер та правила вживання відповідних букв після тих чи інших букв/буквосполучень.
  - Тест на однорідність двійкових послідовностей, скоріш за все, із причин, які є комбінацією причин непроходження двох попередніх тестів.
- Генератор **geffe_generator** не проходить тест на незалежність знаків ймовірно через те, що вхідну послідовність для нього генерують три лінійних регістри із відносно невеликими періодами кожен.