Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з комп’ютерного практикуму №1 з дисципліни
«Математичні основи захисту інформації»

«**Методи формування псевдовипадкових двійкових послідовностей. Методики оцінки якості псевдовипадкових послідовностей**»

Виконав(ла) *ІП-11 Прищепа В. С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Марковський О. П.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Варіант 19

Тип	Розрядність	Об’єм вибірки	Складність
Т(р6)	12+	20000	Л

Ціль роботи: Ознайомитися з сучасними методами генерації псевдовипадкових двійкових послідовностей та методиками оцінки їх якості. Отримати практичні навички комп’ютерної генерації псевдовипадкових двійкових послідовностей.

Хід роботи:

Табличний генератор ПВДП складається з 6 LFSR, розрядність яких відрізняється на одиницю. Поліноміальні функції f(x), використані для оновлення стану LFSR:

12: x^12+x^6+x^4+x+1

13: x^13+x^4+x^3+x+1

14: x^14+x^10+x^6+x+1

15: x^15+x+1

16: x^16+x^12+x^3+x+1

17: x^17+x^3+1

Лістинг:

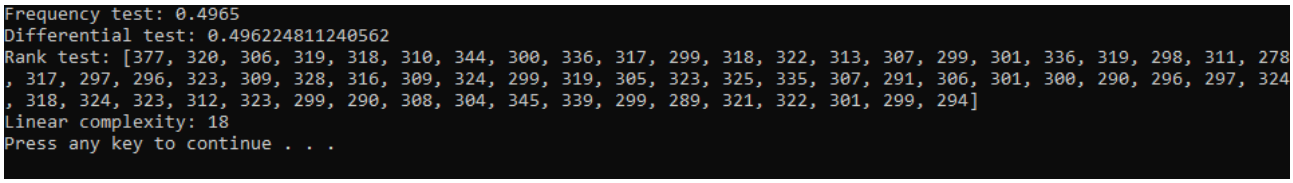
```
import numpy
import copy

# LFSR

def updatestate_12bit(state):
    newbit = (state ^ (state >> 6) ^ (state >> 8) ^ (state >> 11)) & 1
    state = (state >> 1) | (newbit << 11)
    return newbit, state

def updatestate_13bit(state):
    newbit = (state ^ (state >> 9) ^ (state >> 10) ^ (state >> 12)) & 1
    state = (state >> 1) | (newbit << 12)
    return newbit, state

def updatestate_14bit(state):
    newbit = (state ^ (state >> 4) ^ (state >> 8) ^ (state >> 13)) & 1
    state = (state >> 1) | (newbit << 13)
```

```
    return newbit, state

def updatestate_15bit(state):
    newbit = (state ^ (state >> 14)) & 1
    state = (state >> 1) | (newbit << 14)
    return newbit, state

def updatestate_16bit(state):
    newbit = (state ^ (state >> 4) ^ (state >> 13) ^ (state >> 15)) & 1
    state = (state >> 1) | (newbit << 15)
    return newbit, state

def updatestate_17bit(state):
    newbit = (state ^ (state >> 14)) & 1
    state = (state >> 1) | (newbit << 16)
    return newbit, state

# Tests

def frequency_test(sequence):
    n = len(sequence)
    frequency = sum(sequence)/n
    return frequency

def differential_test(sequence):
    n = len(sequence)
    differential = sum(sequence[i]^sequence[i-1] for i in range(1, n))/(n-1)
    return differential

def rank_test(sequence, window):
    n = len(sequence)
    res = [0] * (2**window)
    for i in range(n-window + 1):
        subsequence = sequence[i:i+window]
        index = int(".join(map(str, subsequence)), 2)
        res[index]+=1
    return res

def berlekamp_massey_algorithm(block_data):
    n = len(block_data)
    c = numpy.zeros(n)
    b = numpy.zeros(n)
    c[0], b[0] = 1, 1
    l, m, i = 0, -1, 0
    int_data = [int(el) for el in block_data]
```

```
    while i < n:
        v = int_data[(i - l):i]
        v = v[::-1]
        cc = c[1:l + 1]
        d = (int_data[i] + numpy.dot(v, cc)) % 2
        if d == 1:
            temp = copy.copy(c)
            p = numpy.zeros(n)
            for j in range(0, l):
                if b[j] == 1:
                    p[j + i - m] = 1
            c = (c + p) % 2
            if l <= 0.5 * i:
                l = i + 1 - l
                m = i
                b = temp
        i += 1
    return l

# Initial parameters (key, seed)
key = int("10" * 32, 2)
state_12bit = 1 << 12 | 500
state_13bit = 1 << 13 | 600
state_14bit = 1 << 14 | 700
state_15bit = 1 << 15 | 800
state_16bit = 1 << 16 | 900
state_17bit = 1 << 17 | 1000

# Main cycle (length of generated sequence is 20000 bits)
sequence = []
for i in range(20000):
    # The function returns the first bit of the new LFSR state and writes this state to a
global variable
    feedback_12bit, state_12bit = updatestate_12bit(state_12bit)
    feedback_13bit, state_13bit = updatestate_13bit(state_13bit)
    feedback_14bit, state_14bit = updatestate_14bit(state_14bit)
    feedback_15bit, state_15bit = updatestate_15bit(state_15bit)
    feedback_16bit, state_16bit = updatestate_16bit(state_16bit)
    feedback_17bit, state_17bit = updatestate_17bit(state_17bit)

    # The address of the cell in the table
    address =  (feedback_12bit << 5) | (feedback_13bit << 4) | (feedback_14bit << 3) |
(feedback_15bit << 2) | (feedback_16bit << 1) | feedback_17bit
    key_filter = 1 << address
    generatedbit = (key & key_filter) >> address
```

```
    sequence.append(generatedbit)

# Output of tests` results
sequence_str = ''.join(map(str, sequence))
#print("Generated sequence:\n", sequence_str)
frequency = frequency_test(sequence)
print("Frequency test:", frequency)
differential = differential_test(sequence)
print("Differential test:", differential)
rank = rank_test(sequence_str, 6)
print("Rank test:", rank)
L= berlekamp_massey_algorithm(sequence)
print("Linear complexity:", L)
```

Результати роботи програми:



Результати тестування ПВДП:

1) **Частотний тест**: Значення результату цього тесту становить 0.4965. Це вказує на те, що кількість нулів та одиниць у згенерованій послідовності близька до рівномірного розподілу.

2) **Диференційний тест**: Результат диференційного тесту дорівнює 0.496224811240562. Це значення вказує на те, що різниці між сусідніми бітами у псевдовипадковій послідовності майже рівномірно розподілені.

3) **Ранговий тест**: Відповідно до рангового тесту, у результатах масиву зустрічаються різні значення. Це вказує на різноманітність рангів бітів у послідовності.

4) **Лінійна складність ПВДП**: Результат лінійної складності становить 18. Це означає, що для генерації цієї послідовності можна побудувати лінійний регістр зсуву із 18 бітами.

Висновок: У ході виконання лабораторної роботи №1 було отримано теоретичні та практичні навички роботи з сучасними методами формування псевдовипадкових двійкових послідовностей та методиками оцінки їх якості.

В результаті можна зробити висновок, що згенерована послідовність відповідає вимогам якості псевдовипадкових послідовностей і може бути використана для різноманітних застосувань, що вимагають випадковості.