Звіт до лабоаторної роботи №4

Виконували:
Ракович Дарина ФБ-73
Пекарчук Данило ФБ-74

Мета роботи

Ознайомлення з принципами побудови регістрів зсуву з лінійним зворотним зв'язком; практичне освоєння їх програмної реалізації; дослідження властивостей
лінійних рекурентних послідовностей та їх залежності від властивостей характеристичного полінома регістра

Порядок виконання роботи

0. Уважно прочитати методичні вказівки до виконання комп'ютерного практикуму.
1. Вибрати свій варіант завдання згідно зі списком. Варіанти завдань містяться у
файлі Crypto_CP4 LFSR_Var.
2. За даними характеристичними многочленами p 1 (x), p 2 (x) скласти лінійні рекурентні співвідношення для ЛРЗ, що задаються цими характеристичними многочленами.
3. Написати програми роботи кожного з ЛРЗ L 1 , L 2 .
4. За допомогою цих програм згенерувати імпульсні функції для кожного з ЛРЗ і підрахувати їх періоди.5. За отриманими результатами зробити висновки щодо влавстивостей кожного з
характеристичних многочленів p 1 (x), p 2 (x): многочлен примітивний над F 2 ; не
примітивний, але може бути незвідним; звідний.
6. Для кожної з двох імпульсних функцій обчислити розподіл k-грам на періоді, k≤n i , де n i - степінь полінома f i (x), i=1,2 а також значення функції автокореляції A(d) для
0≤ d ≤ 10. За результатами зробити висновки.

P1(x) = x^20 + x^18 + x^11 + x^10 + x^8 + x^7 + x^6 + x^5 + 1

Період: 1048575
Автокореляція:
d=0: 0,
d=1: 10485760,
d=2: 10485760,
d=3: 10485760,
d=4: 10485760,
d=5: 10485760,
d=6: 10485760,
d=7: 10485760,
d=8: 10485760,
d=9: 10485760,

d=10: 10485760

Таблиця н-грам:

Монограми	Біграми	3-грами	4-грами	5-грами
0: 10485740 1: 10485760	'00': 5242859, '01': 5242880, '10': 5242880, '11': 5242880	'000': 2621418, '001': 2621440, '010': 2621440, '100': 2621440, '101': 2621440, '011': 2621440, '110': 2621440, '111': 2621440	'0000': 1310697, '0001': 1310720, '0010': 1310720, '0100': 1310720, '1000': 1310720, '0101': 1310720, '1010': 1310720, '1011': 1310720, '0110': 1310720, '1100': 1310720, '0011': 1310720, '0111': 1310720, '1110': 1310720, '1101': 1310720, '1001': 1310720, '1111': 1310720	'00000': 655336, '00001': 655360, '00010': 655360, '00100': 655360, '01000': 655360, '10000': 655360, '00101': 655360, '01010': 655360, '10100': 655360, '10101': 655360, '01011': 655360, '10110': 655360, '01100': 655360, '11000': 655360, '00011': 655360, '00110': 655360, '10001': 655360, '00111': 655360, '01110': 655360, '11101': 655360, '11010': 655360, '11001': 655360, '10010': 655360, '01001': 655360, '10011': 655360, '11100': 655360, '01111': 655360, '11110': 655360, '11011': 655360, '01101': 655360, '10111': 655360, '11111': 655360

P(x) = X^24 + X^17 + X^14 + X^13 + X^12 + X^ 9 + X^ 6 + 1

Період: 64897

Автокореляція:

d=0: 0,

d=1: 778844,

d=2: 779024,

d=3: 778740,

d=4: 778592,

d=5: 778848,

d=6: 778632,
d=7: 778740,
d=8: 778592,
d=9: 778788,
d=10: 778752
Таблиця н-грам:

Монограми	Біграми	3-грами	4-грами	5-грами
'0': 779040, '1': 778488	'00': 389615, '01': 389424, '10': 389424, '11': 389064	'000': 194834, '001': 194780, '010': 194700, '100': 194780, '011': 194724, '110': 194724, '111': 194340, '101': 194644	'0000': 97419, '0001': 97414, '0010': 97415, '0100': 97373, '1000': 97414, '1001': 97366, '0011': 97365, '0110': 97454, '1100': 97407, '0111': 97270, '1110': 97270, '1101': 97317, '1010': 97285, '1011': 97359, '0101': 97327, '1111': 97070	'00000': 48739, '00001': 48679, '00010': 48719, '00100': 48720, '01000': 48736, '10000': 48679, '01001': 48637, '10010': 48696, '10011': 48670, '00110': 48759, '01100': 48737, '11000': 48678, '00111': 48606, '01110': 48648, '11100': 48670, '11101': 48600, '11010': 48611, '10100': 48653, '11011': 48706, '10110': 48695, '11001': 48729, '00101': 48695, '01010': 48674, '01011': 48653, '01101': 48717, '10111': 48664, '01111': 48622, '11110': 48622, '10001': 48735, '11111': 48448, '00011': 48695, '10101': 48632

Код програми:

```
from numba import cuda, jit, njit, vectorize, types
from numba.typed import List, Dict
import numpy as np
import random
from collections import defaultdict
import asyncio

@njit()
def lfsr(init_state, curr_state, polynom, autocorr, period):
	counter = 0
	prev_list = List()
	while True:

	for j in range(curr_state.size):
	prev_list.append(curr_state[j])
	for d in range(11):
		autocorr[d] += curr_state[j] ^ curr_state[(j+d) % curr_state.size]
		# print(autocorr[d])

	new_bit = 0
	for item in polynom:
	if item != init_state.size:
		new_bit ^= curr_state[item]

	counter += 1
	if counter > curr_state.size:
	print(counter)

	for j in range(curr_state.size-1):
	curr_state[j] = curr_state[j+1]
	curr_state[-1] = new_bit

	validated = True
	for j in range(curr_state.size):
	if curr_state[j] != init_state[j]:
		validated = False
		break

	if validated:
	period[0] = counter
```

```
        return prev_list

        return None


async def get_polygrams(prev, num, out_file):
        result = defaultdict(int)
        poly_counter = 0
        poly_string = ''

        for item in prev:
        poly_string += str(item)
        if len(poly_string) == num:
        result[poly_string] += 1
        poly_string = poly_string[1::1]
        poly_counter -= -1 # oh yeah baby

        message = f"{num}-gramm: total count = {poly_counter}: \n{result}\n\n"
        print(message)
        out_file.write(message)
        return poly_counter


def runner(polynom, out_file):

        initial_message = f"Polynom: {polynom}\nq ** (n) - 1: {2 ** (polynom[0]) - 1}"
        print(initial_message)
        out_file.write(initial_message)

        init_state = [1] + [0] * (polynom[0] - 1)
        for i in range(polynom[0]):
        if i < polynom[0] - 1:
        init_state = [0] + init_state[:-1]
        continue

        initial_array = np.array(init_state, np.int64)
        polynom_array = np.array(polynom, np.int64)
        current_array = np.copy(initial_array)
        autocorr_dict = Dict.empty(key_type=types.int64, value_type=types.int64)

        for i in range(0, 11):
        # will fill up later
```

```
        autocorr_dict[i] = 0

        period = np.ones(1, dtype=np.int64)
        res = lfsr(  # linear feedback shift register runs here
        initial_array,
        current_array,
        polynom_array,
        autocorr_dict,
        period
        )

        message = (
        f"\nInitial state: {''.join([str(i) for i in initial_array])}\n"
        f"period: {period[0]}\n"
        f"result state: {''.join([str(i) for i in current_array])}\n"
        f"autocorrection values: {autocorr_dict}\n" + "="*40 + "\n"
        )
        print(message)
        out_file.write(message)

        # running getting polygrams in async
        loop = asyncio.get_event_loop()
        futures = list()
        for i in range(1, 6):
        futures.append(get_polygrams(res, i, out_file))

        results = loop.run_until_complete(asyncio.gather(*futures))
        loop.close()

        for result in results:
        print(result)

        init_state = [0] + init_state[:-1]

        return


def main():
        polynoms = [
        (20,18,11,10,8,7,6,5,0),
        (24,17,14,13,12,9,6,0)
        ]
```

```
        for i in range(0, len(polynoms)):
        filename = f"out{i}.txt"
        out_file = open(filename, "w")
        runner(polynoms[i], out_file)
        out_file.close()


if __name__ == "__main__":
        main()
```