Politechnika Białostocka

Sprawozdanie z pracowni specjalistycznej

Sztuczna inteligencja

Ćwiczenie numer: 4-5

Temat: **Generatory liczb pseudolosowych. Szyfry strumieniowe.**

Wykonujący ćwiczenie: **Michał Mitrosz**

Studia dzienne

Kierunek: Informatyka

Semestr: VI

Grupa zajęciowa: PS 4

Prowadzący ćwiczenie: mgr inż. Dariusz Jankowski

Data wykonania ćwiczenia:
22.05.2021 r.

1. Realizacja zadań

1. Zaimplementuj generator liczb pseudolosowych bazujący na LFSR o zadanym stopniu wielomianu.

```
class LFSR:
    def __init__(self, start=1, bits=8, taps=[1, 2, 3, 7]):
        self.bits = bits
        self.taps = taps
        self.state = start


    def set(self, state):
        self.state = state


    def get(self):
        return self.state


    def get_xored_taps(self):
        xored = 0

        for i in self.taps:
            xored ^= (self.state >> i) % 2

        return xored


    def next(self, new_bit = -1):
        if new_bit == -1:
            new_bit = self.get_xored_taps()

        self.state <<= 1
        self.state %= 2**self.bits
        self.state |= new_bit

        return self.state
```

Rys. 1 - Kod generatora bazującego na LFSR

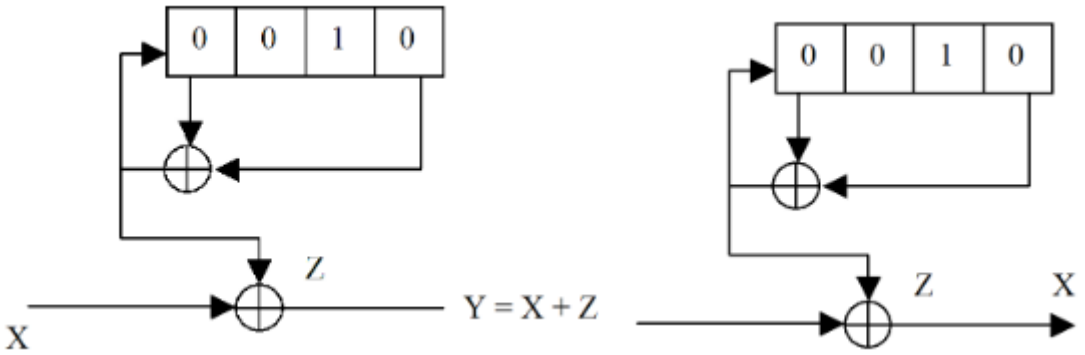
```
lfsr = LFSR()
print(lfsr.get())

for i in range(20):
    print(lfsr.next())
```

Rys. 2 - Wygenerowanie pierwszych 20 wartości dla domyślnych wartości początkowych

```
1
2
5
11
22
44
88
177
99
199
143
30
61
122
244
232
208
161
67
135
15
```

Rys. 3 - Wygenerowane wartości, które pokrywają się z rzeczywistym LFSR

2. Zaimplementuj kryptosystem bazujący na schemacie Synchronous Stream Cipher dla podanego wielomianu i ziarna.



Rys. 4 - Schemat Synchronous Stream Cipher

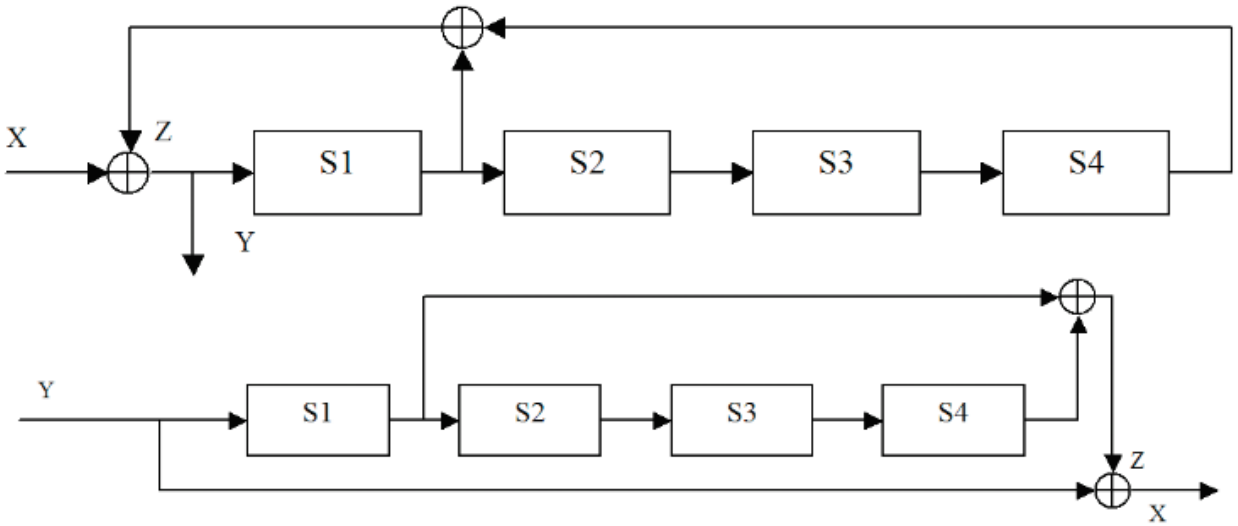
```python
def ssc(message, seed, mask):
    taps = []
    for index, item in enumerate(mask):
        if item == '1':
            taps.append(index)

    lfsr = LFSR(int(seed[::-1], 2), len(mask), taps)
    ret = 0

    for i in message:
        ret *= 2
        ret += int(i) ^ (lfsr.next()%2)

    return bin(ret)[2:]
```

Rys. 5 - Kod źródłowy programu realizującego zadanie

```python
def test_ssc(self):
    self.assertEqual(ssc('11101001', '0010', '1001'), '10010011')
```

Rys. 6 - Test do programu

3. Zaimplementuj kryptosystem bazujący na schemacie Ciphertext Autokey dla podanego wielomianu i ziarna.



Rys. 7 - Schemat Ciphertext Autokey

```python
def autokey(message, seed, mask, encode=True):
    taps = []
    for index, item in enumerate(mask):
        if item == '1':
            taps.append(index)

    lfsr = LFSR(int(seed[::-1], 2), len(mask), taps)
    ret = 0

    for i in message:
        bit = lfsr.get_xored_taps() ^ int(i)
        lfsr.next(bit if encode else int(i))
        ret *= 2
        ret += bit

    return bin(ret)[2:].zfill(len(message))
```

Rys. 8 - Kod źródłowy programu realizującego zadanie

W podanej funkcji, można przełączać się pomiędzy szyfracją a deszyfracją podając odpowiedni parameter encode, odpowiednio True i False.

```python
def test_autokey(self):
    self.assertEqual(autokey('11101001', '0011', '1001', encode=True), '00110011')
    self.assertEqual(autokey('00110011', '0011', '1001', encode=False), '11101001')
```

Rys. 9 - Testy dla szyfracji i deszyfracji

2. Wnioski

Wszystkie zadania udało się zrealizować bez większych problemów, wszystkie testy zaprezentowane na zrzutach zwróciły oczekiwany rezultat.