Introduction to LFSRs

In the following we consider LFSRs on $\mathbb{F}_2$.

Question 1. *Write a function* `LFSR_step(P, state)` *that simulates one step of an LFSR. It takes as input a retroaction polynomial $P \in \mathbb{F}_2[X]$ (represented as a list of bits) and the state S_t (represented as a list of bits), and returns the state at time $t+1$ and the output at time t.*

(Recall that the XOR operator in Python is `^`*.)*

Question 2. *Write a function* `LFSR(P, state, N)` *that takes a retroaction polynomial, the initial state S_0 and an integer N, and returns the N first elements of the output of the LFSR of retroaction P, initialized with S_0.*

Question 3. *Check that the LFSR of length 5 and retroaction polynomial $P(X) = 1 + X^4 + X^5 \in \mathbb{F}_2[X]$, initialized by $S_0 = (0, 0, 1, 0, 1)$, returns a sequence starting with $0, 0, 1, 0, 1, 0, 1, \ldots$. What is its period?*

Cycles

Question 4. *Write a function that takes the initial state S_0 of an LFSR and its polynomial P, and returns the set of all different internal states $S_0, S_1, \ldots, S_{T-1}$ obtained on a period.*

Question 5. *Test with $P(X) = 1 + X^4 + X^5 \in \mathbb{F}_2[X]$ and the initial states $(0, 0, 1, 0, 1)$, $(0, 1, 0, 0, 0)$, $(1, 0, 1, 0, 0)$. What do you notice?*

Question 6. *Write a function taking as input the retroaction polynomial and returning all the possible cycles (by making the initial state vary).*

Question 7. *How many cycles are produced by the LFSR of length 5 and retroaction polynomial P? Same question with the LFSR of length 4 and polynomials:*

- $1 + X^2 + X^4$
- $1 + X + X^2 + X^3 + X^4$
- $1 + X + X^4$

The Berlekamp-Massey Algorithm

LFSRs are very fast, they output sequences with very large periods, and good statistical properties. However they are not enough to make secure stream ciphers: indeed the sequence generated by an LFSR is predictible.

Question 8. *Let $(s_n)_{n\in\mathbb{N}}$ be the sequence generated by an LFSR. Let $\Lambda(s)$ be its linear complexity (the degree of the minimal polynomial).*

Show that $2\Lambda(s)$ consecutive terms of the sequence are sufficient to characterize s. More precisely, show that if we know $\Lambda(s)$, as well as $2\Lambda(s)$ consecutive terms of s, then we can compute the minimal polynomial P.

Indication: *write the coefficients of P as the unknowns in a linear system.*

Question 9. *Suppose that we know an upper bound $\Lambda(s) \leq \ell$. Deduce an algorithm that takes s as input and returns $\Lambda(s)$, as well as the retroaction polynomial, in time $\mathcal{O}(\ell^4)$.*

Algorithm 1 Berlekamp-Massey algorithm.

```
     Input:  sequence $\mathbf{s} = (s_0, \ldots, s_{n-1})$ of elements in $\mathbb{F}_q$
     Output:  $\Lambda(\mathbf{s})$ and $P$, the minimal retroaction polynomial
 1: $P(X) \leftarrow 1$
 2: $Q(X) \leftarrow 1$
 3: $\Lambda \leftarrow 0$
 4: $m \leftarrow -1$
 5: $d' \leftarrow 1$
 6: for $t \in \{0, \ldots, n-1\}$ do
 7:     $d \leftarrow s_t + \sum_{i=1}^{\Lambda} p_i s_{t-i}$
 8:     if $d \neq 0$ then
 9:         $T(X) \leftarrow P(X)$
10:         $P(X) \leftarrow P(X) - d(d')^{-1}Q(X)X^{t-m}$
11:         if $2\Lambda \leq t$ then
12:             $\Lambda \leftarrow t + 1 - \Lambda$
13:             $m \leftarrow t$
14:             $Q(X) \leftarrow T(X)$
15:             $d' \leftarrow d$
16:         end if
17:     end if
18: end for
19: Return $\Lambda$ and $P(X)$
```

This is a polynomial algorithm already, but not very efficient in practice. We can do much better using the Berlekamp-Massey algorithm.

Question 10. *Implement the Berlekamp-Massey algorithm for $\mathbb{F}_2$. Several important points to notice:*

- *Because we are in $\mathbb{F}_2$, d' is always equal to 1. The Line 10 is simplified.*
- *You can represent the polynomials P and Q as lists of binary coefficients, that you can initialize by* `[1] + [0]*len(s)` *as the list will never be bigger.*
- *The operation at Line 10 becomes an operation on lists. In my code it's:*

```
for i in range(len(P) - (t-m)):
    P[i + t-m] = P[i + t-m] ^ Q[i]
```

What is the complexity of the algorithm? Test your implementation on an LFSR sequence generated by your function `LFSR` *of Question 2, and observe that the result is incorrect if the input sequence is too small.*

The idea of the algorithm is that the value d computed during the loop is telling us whether the current polynomial P looks good or not. If d is zero, we have nothing to do. If d is not zero, we adjust P so that a recalculation of d would give 0. Thus, at time t, the algorithm determines an LFSR of minimal length that produces the t first values of the sequence.