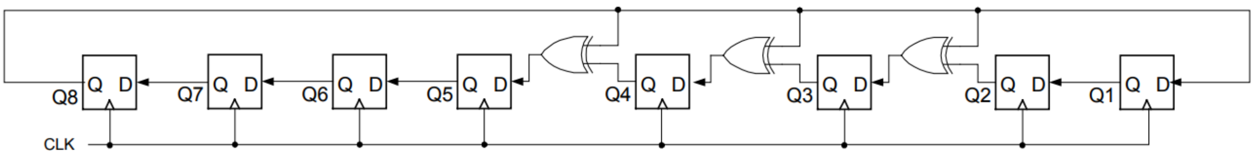
密碼工程quiz4

problem1

(a)

Yes, $x^8+x^4+x^3+x^2+1$ is a primitive polynomial.
Theorem: **Let c(x) be a characteristic polynomial of an LFSR of length n. Then c(x) is primitive polynomial iff every non-zero initial state of an LFSR produces a pseudorandom sequence of length 2^n-1.**

First of all, we use $x^8+x^4+x^3+x^2+1$ to build an 8-bit LFSR. (like the circuit below)



Next, we start from the initial non-zero bit pattern: 00000001.
If LFSR can generate 255 (2^8-1) different non-zero bit pattern, then it can support that $x^8+x^4+x^3+x^2+1$ is a primitive polynomial.
I use python code to test. As a result, $x^8+x^4+x^3+x^2+1$ is a primitive polynomial.

```
密碼工程\HW\HW4\problem1.py"
There are 255 different non-zero bit pattern
x^8 + x^4 + x^3 + x^2 + 1 is a  primitive polynomial
```

(b)

The maximum cycle length is 2^m - 1, and m is the highest degree of the polynomial.
In this problem, m=8.
Therefore, The answer is: 2^8 - 1 = 255.

(c)

No, not all irreducible polynomials are primitive polynomials.

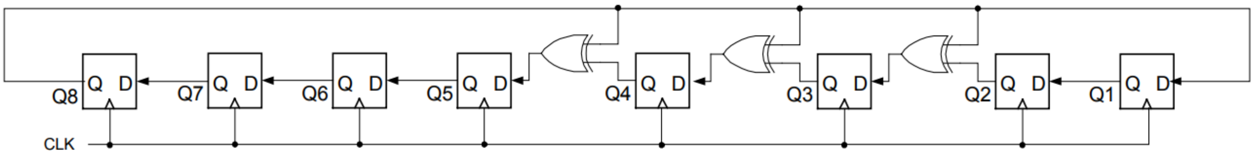
An irreducible polynomial is a polynomial that cannot be factored into the product of two non-constant polynomials over the same field. However, a primitive polynomial is a specific type of irreducible polynomial that generates a primitive element in a finite field extension.

In other words, while **all primitive polynomials are irreducible**, **not all irreducible polynomials are primitive**. Primitive polynomials have an additional property of generating a primitive element in a field extension, which not all irreducible polynomials possess.

problem2

(a)

I use $x^8 + x^4 + x^3 + x^2 + 1$ as a characteristic polynomial to buld a LFSR (like the circuit below) to help me to generate the key.

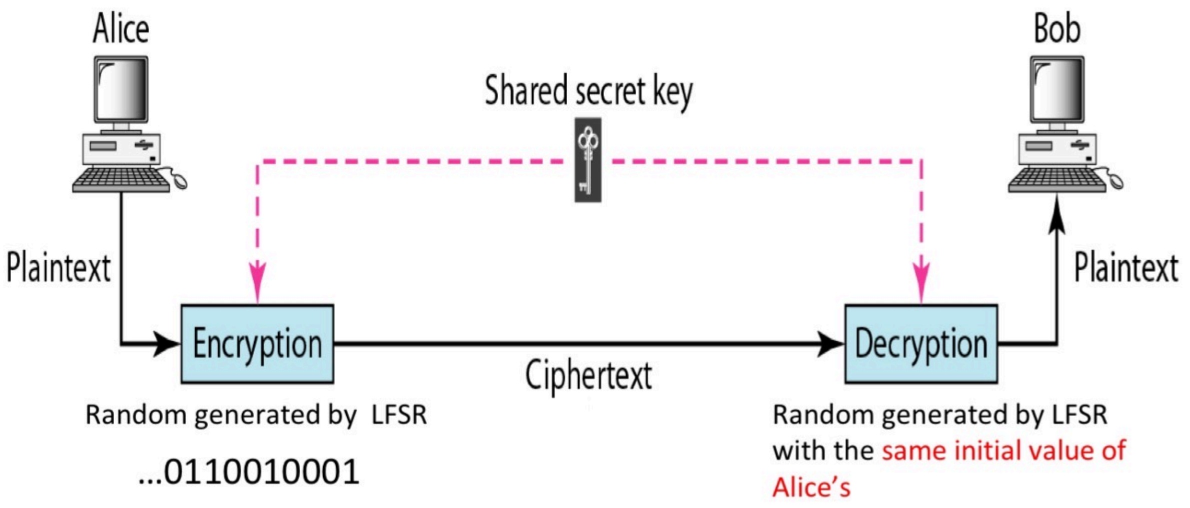


Next, convert plaintext to binary, and then calculate the length of it, which is also the length of the key.
We start from the initial state 00000001. Let the first bit stored in key. Then, make LSFR generate the next state. Repeat the process untill the length of key is enough.
Finally, we can use the key to encrypt the plaintext.
ciphertext in binary and ciphertext:

```
cipertext_binary:
0100000001001000000000101110110001101000100111011000101100001111000011000001100111011001110100110100001010101111111100011001
1011111100001000000110111000100001101010100011101101011010110011011011110010110100111100010010100010110100011110101001011
0000101010011010101000111110111000101010100000110110110001000110110111110001011111010100000101011100110111111111100000101100
1001010000000100000010101011101010000000100110110011000101010101011100110100110011001011011011100111111111110110000110111 0
101101100001111101100110010111010011010101100111010100000000000001111110101000000000111110001100011100100010001000100110000 0
0010000110000001111001101100100110010101100111110111000111010111100001000110111100011001010110110001100111100000100010110
1111000000111010100111111111001001110100010110110010000100000100100111001011001000100101100010010101100010010100001111110 0
1011100011110110110011100000110001100011010000111000010001011000100011001000110000111100000000111101001100000100100101011 00
111111110001001111011010101100100010111110010101010001101000110111110100010101010110010110101011010110111110000111011111 01
0100010011101000011000010001001001011110110001110111011111001111011111100011101101010101000101000111010110110111111100011001
0101100111010011100111101000011110111000011110110000111101000110101111001000000100100011001010000100010001011011010100010 01
0101101101011111011001011011110011011001001110011110100010011000100010011001011001111011001000100111010110101101000111110 1
0001011011001110111111111101110100000101000111000110101111010011000000001000100110101011110110010001100001101010011010111 00
0100111000000100001000000111000111011110010000000000101100100010100000000011010110010111011010100001001011110101011101001011
0100001000101110101001010001000111001011011011010001101111110010001111000101011011001110011110110001110010100100011000110 01
0101100001111000011110000001110010010110011001110111011110011010001100001101100111110010001111011000001011111101110001100 10
0011111010110101100110010000111001001111111111111010100101001100101110100100001111000010010101010101010000001110110000101 1
0000000010101100000000000000111100010000000101110101110001011101010010111010000101101101001101011110111100000011011101100000 1
10001010100110100101101110101101001111011000010001000001111010000100010001111011010001101010010111000100101101101010110001
000101011101110100100100001100001111110101111110111

cipertext:
@HØÑ;3³¦_ÆoÂâ£µ¬ÛËOG©aST}ÅPmÛâú¹¿ðZP UÔÙ«f[sÿØn¶f]5gPý@>1`y²egÜuðã+c<o©ÿ'E²IË"XCòãÛ81b20ð$-ÿÚ²/FôUe«[áßQ:D D
±Ýóßª:Ûñ9è{°ôkÈ$j%m}ódç¢b&YìÖ´}Îÿº
8×¦W²0Ô×8ï
"5jõt´"êQ¶Ñ¾GÙÏceaáàä³;¼ÑÏìî2>µOÿ©LºCÂUTÂÀ+.¸ºBÚkÞì©¥ºÓØDG´j\KjÄWtÃõ÷
```

Subsequently, we start to decrypt the ciphertext. As professor taught in the class, we can know that the key of encryption and the key of decryption is the same.



Therefore, we use the key, which we generated before, to decrypt.
plaintext in binary and plaintext:

```
plaintext_binary:
010000010101010001001110010110010100001101010101010101110100010101000000101010010010001010101001101010100010100100100100101
010110010010010100111001000111010101000100111101000010010001010100000010100011101010010010001010100000101010100010101010100
111001001001010101100100010101010010010100110100100101010100010110010101010001001000010000010101010001010100010100100100000
010100111001010011010000110100010101001110010001000101001101000100010010010101001101000011010010010101000000100110001001001
010011100100000101010010010110010100010001001001010101100100100101000100010001010101001101010100010011110101001101001111101
001100010101100100010101010100010010000100010101001001010011100100001101010010010001010100000010101001101001001010011100100
011101001100010110010100001101001111010011010101000000100110001000101010110000101000000101001001001111010000100100110001000 1
010100110101010011010101000100100001000001010101000101010001001000010001010101011101001111010100100100110001000100010001000110
010000010100001101000101010100110101011101000101010101110100100101001100010011000100001101001111010011100101010001001001001 01
001110010101010100010101010100010011110100001001000101010001110101010101001001010001000100010101000100010000100101100101 01
010001001000010001010100100101000100010001010100000101010100010010000100000101010100010101110100010101000011010000001010011
100100000101000011010010000100100101000101010110010001010100110100111101001101010001010101000100100001001001010101110
010001110100110101010101010000110100100001000111010100100100010101000000101010100010001010101001001010100010011110100011101
000101010101000100100001000101010100100101010001001000010000010100111001010111010001010100001101000001010011100100100100 100
111001000100010010010101011001001001010001000101010101000001010011000100110001011001010000001010001100101010001000101010100
1001000000101001100010011000101010001001000010000001010101000101011101000001010100110101010001001000010001010100100101000100
010001010100000010101010001001000010000001010101000100110001000101010001000101010001001111010101000100100001000101010000 1101
010010010000101010000001010101000100100101001111010011100100111101000110010011110101010101010010010101010100111001001001 0 101
011001000101010100100101001101001001010101000101100101001001010011100101010001001000010001010100011001001001010100100 10100
110101010001010000001001100010000001010000110100 0101
plaintext:
ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRANSCENDSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCOMPLEXPROBLEMSTHATTHEWORLDF
ACESWEWILLCONTINUETOBEGUIDEDBYTHEIDEATHATWECANACHIEVESOMETHINGMUCHGREATERTOGETHERTHANWECANINDIVIDUALLYAFTERALLTHATWASTHEID
EATHATLEDTOTHECREATIONOFOURUNIVERSITYINTHEFIRSTPLACE
```

➜ We can observe that the plaintext remains unchanged from the original. Therefore, my encryption is correct.

(b)

Starting from the first plaintext letter, we deduce the first bit of the initial state by performing an exclusive OR operation with 0 and the corresponding bit in the ciphertext. Moving forward to next letter, we determine the first bit of the 8th state by again performing an exclusive OR operation with 0 and the corresponding bit in the ciphertext. This process repeats until all plaintext letters are processed. Consequently, we observe that every 255 bits discovered constitute one cycle, implying that 255 states create a cycle. Consequently, we can rearrange these 255 bits to form a key.

state :

0 → 8 → 16 → ... → 248 → 1 → 9 → 17 → ...

→ 249 → 2 .....

⇒ We can find that 255 states form a cycle.

Since 255 bits can form a cycle, we can know that the length of LFSR is 8 (∵ 2^8-1 = 255). Then, we can list the equtation:
(a means the bit of key, and c is the parameter we want to solve)

$a_n = (a_{n+1}C_7 + a_{n+2}C_6 + a_{n+3}C_5 + a_{n+4}C_4 + a_{n+5}C_3 + a_{n+6}C_2 + a_{n+7}C_1 + a_{n+8}C_0) \bmod 2$

$a_{n+1} = (a_{n+2}C_7 + a_{n+3}C_6 + a_{n+4}C_5 + a_{n+5}C_4 + a_{n+6}C_3 + a_{n+7}C_2 + a_{n+8}C_1 + a_{n+9}C_0) \bmod 2$

.

.

.

.

$a_{n+7} = (a_{n+8}C_7 + a_{n+9}C_6 + a_{n+10}C_5 + a_{n+11}C_4 + a_{n+12}C_3 + a_{n+13}C_2 + a_{n+14}C_1 + a_{n+15}C_0) \bmod 2$

Once we solve the equation, then we can know the characteristic polynomial.

problem3

(a)

I import **random** in my python code.
The code of Naive algorithm function:

```
def naive_algorithm():
    cards = [1, 2, 3, 4]
    for i in range(4):
        n = random.randrange(0, 4)
        cards[i], cards[n] = cards[n], cards[i]
    return cards
```

➜ We iterate index i in ascending order. Swap index i and the random position n, which is chosen from between 0 and 3, in each iteration.

The code of fisher-yates shuffle function:

```
def fisher_yates_shuffle():
    cards = [1, 2, 3, 4]
    for i in range(3, 0, -1):
        n = random.randrange(0, i+1)
        cards[i], cards[n] = cards[n], cards[i]
    return cards
```

➜ We iterate index i in descending order. Swap index i and the random position n, which is chosen from between 0 and i, in each iteration.

The result of Naive algorithm:

```
Naive algorithm:
(1, 2, 3, 4) : 38967
(1, 2, 4, 3) : 39302
(1, 3, 2, 4) : 39129
(1, 3, 4, 2) : 55340
(1, 4, 2, 3) : 43101
(1, 4, 3, 2) : 34918
(2, 1, 3, 4) : 39412
(2, 1, 4, 3) : 58277
(2, 3, 1, 4) : 54776
(2, 3, 4, 1) : 54657
(2, 4, 1, 3) : 43004
(2, 4, 3, 1) : 43064
(3, 1, 2, 4) : 42845
(3, 1, 4, 2) : 42744
(3, 2, 1, 4) : 35207
(3, 2, 4, 1) : 42969
(3, 4, 1, 2) : 43368
(3, 4, 2, 1) : 38990
(4, 1, 2, 3) : 31047
(4, 1, 3, 2) : 35377
(4, 2, 1, 3) : 35013
(4, 2, 3, 1) : 30947
(4, 3, 1, 2) : 38726
(4, 3, 2, 1) : 38820
```

The result of Fisher–Yates shuffle:

```
Fisher-Yates shuffle:
(1, 2, 3, 4) : 41384
(1, 2, 4, 3) : 41756
(1, 3, 2, 4) : 41769
(1, 3, 4, 2) : 41480
(1, 4, 2, 3) : 41724
(1, 4, 3, 2) : 41861
(2, 1, 3, 4) : 41781
(2, 1, 4, 3) : 41564
(2, 3, 1, 4) : 41618
(2, 3, 4, 1) : 41500
(2, 4, 1, 3) : 41870
(2, 4, 3, 1) : 41371
(3, 1, 2, 4) : 41712
(3, 1, 4, 2) : 41699
(3, 2, 1, 4) : 42021
(3, 2, 4, 1) : 41569
(3, 4, 1, 2) : 41528
(3, 4, 2, 1) : 41628
(4, 1, 2, 3) : 41465
(4, 1, 3, 2) : 41978
(4, 2, 1, 3) : 41223
(4, 2, 3, 1) : 41808
(4, 3, 1, 2) : 41566
(4, 3, 2, 1) : 42125
```

(b)

From the result, we can find that the Fisher-Yates shuffle generates permutations with a more evenly distributed range of values, while the naive algorithm appears to have a narrower range of permutation values for the same input sequence. Therefore, Fisher-Yates shuffle is better one, which can generate the true randomness.

(c)

Depending on how randomness is generated and swaps are performed, the naive algorithm may introduce biases. It will make the result not as random or uniformly distributed as desired, particularly when dealing with larger lists.