# Lab11 Solution

YAO ZHAO

# Lab11 Q1

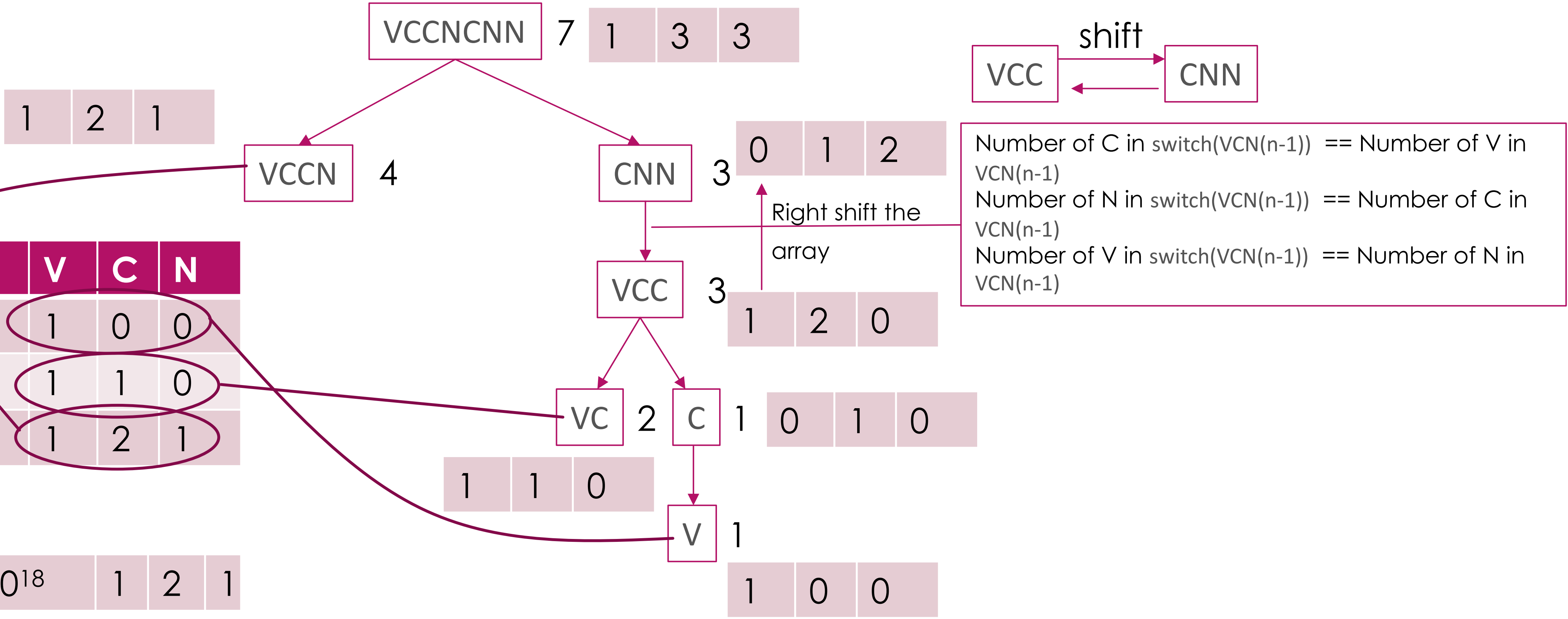
VCN(1) = VC
VCN(2) = VCCN
VCN(3) = VCCNCNNV
...
VCN(n) = VCN(n−1)+switch(VCN(n-1))

switch(s) means changing all 'V' to 'C', all 'C' to 'N', and all 'N' to 'V' in string s

| VCN(n) | len | V | C | N |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 2 | 1 | 1 | 0 |
| 2 | 4 | 1+0= 1 | 1+1= 2 | 1+0=1 |
| 3 | 8 | 1+1 =2 | 1+2 = 3 | 2+1 = 3 |
| n | $2^n$ | V(n) = V(n-1)+N(n-1) | C(n)=C(n-1)+V(n-1) | N(n)=N(n-1)+C(n-1) |

# Lab11 Q1 Sample

Sample:
how many ‘V’, ‘C’, and ‘N’ are there in the string from the 1st position to the 7th position.




| VCN(n) | len | V | C | N |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 2 | 1 | 1 | 0 |
| 2 | 4 | 1 | 2 | 1 |

| | | | | |
|---|---|---|---|---|
| 60 | $1.15...*10^{18}$ | 1 | 2 | 1 |

# Lab11 Q1 Pseudo-code

Generate table $T$ according Page.2

*(V,C,N)* **Count_VCN**($L$){

if ($L$==0) return *(0,0,0)*

$index = \lfloor log_2 L \rfloor$

*(V1, C1, N1)* ← get the value from the table $T$ according the $index$

(*V2, C2, N2)* ← **Count_VCN**($L$ -$2^{index}$)

return ($V1+N2, C1+V2, N1+C2$)

}

# Lab11 Q2

4 11
4 5 1 4

A1 A2 A3 A4

| S | | | |
|---|---|---|---|
| 0 | {}->0 | 7 | {}->0 |
| 1 | {A3*A3}->1 | 8 | {}->0 |
| 2 | {}->0 | 9 | {A1*A2,A2*A1,A2*A4,A4*A2}->4 |
| 3 | {A2*A2}->1 | 10 | {}->0 |
| 4 | {A1*A3, A3*A1, A3*A4, A4*A3}->4 | | |
| 5 | {A1*A1, A2*A3, A3*A2, A4*A4, A1*A4,A4*A1}->6 | | |
| 6 | {}->0 | | |

# Lab12 Q2

4 11
4 5 1 4

2 is the primitive root of 11, we have the following formulas:

$$
\begin{aligned}
2^1 &\equiv 2(mod\ 11)\\
2^2 &\equiv 4(mod\ 11)\\
2^3 &\equiv 8(mod\ 11)\\
2^4 &\equiv 5(mod\ 11)\\
2^5 &\equiv 10(mod\ 11)\\
2^6 &\equiv 9(mod\ 11)\\
2^7 &\equiv 7(mod\ 11)\\
2^8 &\equiv 3(mod\ 11)\\
2^9 &\equiv 6(mod\ 11)\\
2^{10} &\equiv 1(mod\ 11)\\
2^{10+1} &\equiv 2(mod\ 11)\\
&\dots
\end{aligned}
$$

A1 = $4(mod\ 11) = 2^2(mod\ 11)$
A2 = $5(mod\ 11) = 2^4(mod\ 11)$
A1*A2 = $2^2 * 2^4(mod\ 11) = 2^6 \equiv 9(mod\ 11)$

| exponent | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| remainder | 2 | 4 | 8 | 5 | 10 | 9 | 7 | 3 | 6 | 1 |

| remainder | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| exponent | 10 | 1 | 8 | 2 | 4 | 9 | 7 | 3 | 6 | 1 |

## Convert this problem to a Polynomial Multiplication Problem

4 5 1 4

$2^2 \;\; 2^4 \;\; 2^{10} \;\; 2^2$

$$(2*2^2+\ 2^4+2^{10})\ *(2*2^2+\ 2^4+2^{10})$$

Polynomial Multiplication

$$(4*2^4+\ 4*2^6+1*2^8+4*2^{12}+2*\ 2^{14}+1*2^{20})$$

P =11

| remainder | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| exponent | 10 | 1 | 8 | 2 | 4 | 9 | 7 | 3 | 6 | 1 |

When the remainder is 0, special handling is required.

| exponent | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| remainder | 2 | 4 | 8 | 5 | 10 | 9 | 7 | 3 | 6 | 1 |

| remainder | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| number | | 1 | | 1 | 4 | 4+2 | | | | 4 | |

# Lab11 Q2 Pseudo-code

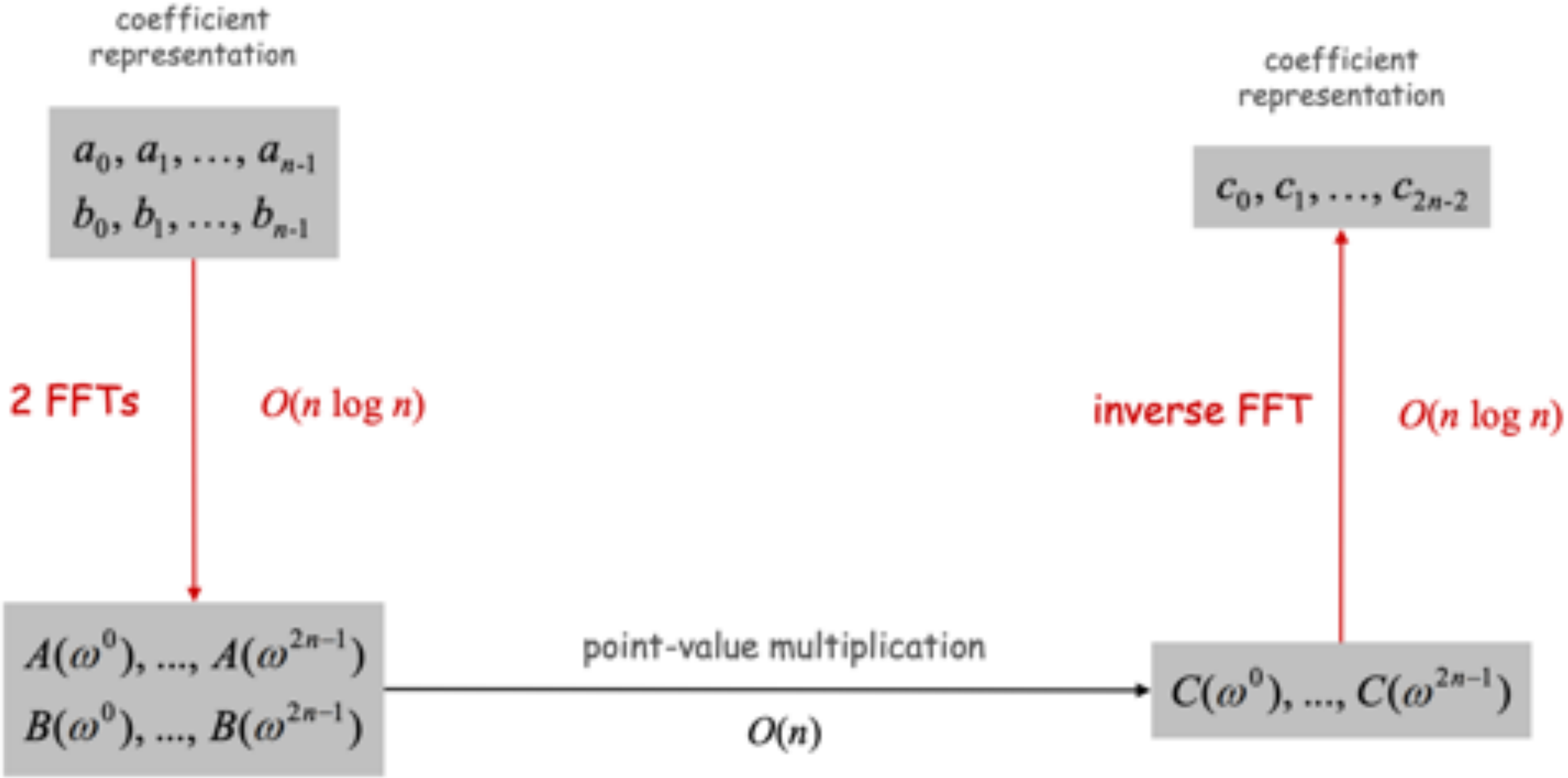
Input parameters: a prime P and a sequence A: $A_1, A_2, A_3 \ldots A_n$
Output parameter: a sequence S: $S_0, S_1, S_2 \ldots S_{P-1}$

```
S FaFaFa(P, A){
    calculate the primitive root g of P
    calculate g^i % P (1≤i≤P-1)to generate the list exponent_to_remainder and remainder_to_ exponent
    initial the coefficient sequence B: B_0, B_1, B_2…B_{P-1} to 0
    for k = 1 to n{
        e← get exponent from A_k%P according the list remainder_to_ exponent
        B_e← B_e+1
    }
    C_2…C_{2P-2} ←Polynomial_Multiplication(B_1, B_2…B_{P-1} )
    initial the coefficient sequence S: S_0, S_1, S_2…S_{P-1} to 0
    for e = 2 to 2P -2{
        r ←get remainder from e%(P-1) according the list exponent_to_remainder
        S_r ←S_r + C_e
    }
    calculate S_0
    return S
}
```

# Polynomial Multiplication

**Theorem.** Can multiply two degree $n$-1 polynomials in $O(n \log n)$ steps.

pad with 0s to make $n$ a power of 2



# FFT Pseudo-code

input： n, $a_0$, $a_1$,..., $a_{n-1}$（n = 2$^k$ (k = 0,1,2… )   you can check n & (n - 1) == 0）
output： $y_0$, $y_1$,..., $y_{n-1}$

FFT(n, $a_0$, $a_1$,..., $a_{n-1}$) {
  if (n == 1) return $a_0$
    ($e_0$, $e_1$,..., $e_{\frac{n}{2}-1}$)<=FFT(n/2, $a_0$, $a_2$,..., $a_{n-2}$)
    ($d_0$, $d_1$,..., $d_{\frac{n}{2}-1}$)<=FFT(n/2, $a_1$, $a_3$,..., $a_{n-1}$)
    for k = 0 to n/2 - 1 {
      $\omega^k = e^{-2\pi ik/n}$ //When you write your code, you can use:$\omega^k$=$\cos(-\frac{2\pi k}{n}) + \mathrm{i}\sin(-\frac{2\pi k}{n})$

      $y_k = e_k + \omega^{k*} d_k$
      $y_{k+\frac{n}{2}} = e_k - \omega^{k*} d_k$
  }
  return ($y_0$, $y_1$,..., $y_{n-1}$)
}

# IFFT Pseudo-code

input：  n, $a_0$, $a_1$,…, $a_{n-1}$ （n = $2^k$ (k = 0,1,2… )   you can check n & (n - 1) == 0）
output：  $y_0$, $y_1$,…, $y_{n-1}$

```
IFFT(n, a_0, a_1,…, a_{n-1}) {
    if (n == 1) return a0
        (e_0, e_1,…, e_{n/2-1})<=IFFT(n/2, a_0, a_2,…, a_{n-2})
        (d_0, d_1,…, d_{n/2-1})<=IFFT(n/2, a_1, a_3,…, a_{n-1})
        for k = 0 to n/2 - 1 {
            ω^k = e^{2πik/n}  // When you write your code, you can use:ω^k=cos(2πk/n) + isin(2πk/n)

            y_k = (e_k + ω^k*d_k)/n
            y_{k+n/2} = (e_k − ω^k*d_k)/n
    }
   return (y_0, y_1,…, y_{n-1})
}
```