POLYNOMIALS

We will consider two types of complexity:

- Arithmetic complexity: number of arithmetic operations and assignments done by an algorithm A. This works essentially when we perform iterations on finite sets or finite fields. But if we perform operations on $\mathbb{Q}$ then we cannot consider it.

- Binary complexity: It is appropriate to "decompose" the integers in a base B which is (in practice) a fixed power of 2. Each integer is seen in binary notation or we can consider it as an element in $\{0, \dots, B-1\}$. We say that binary complexity is the arithmetic complexity associated with $A = \{0, \dots, B-1\}$, with operations $+, \times \bmod B$, comparisons, ...

→ size in terms of BITS of the input of the algorithm

Multiplication of polynomials

Fast algorithms for multiplying polynomials and integers are important for efficient algorithms in computer algebra. Most of the gain in their complexity rely on the efficiency of the multiplication.

Let R be a ring. Define the polynomial ring $R[x] = \{f_0 + f_1 x + \dots + f_m x^m \mid f_i \in R\}$ with $+$ and $\cdot$, defined as

$$f + g = \sum (f_i + g_i) x^i$$

$$f \cdot g = \sum h_k x^k \quad \text{with} \quad h_k = \sum_{i+j=k} f_i g_i$$

$$\deg f = \max\{i : f_i \neq 0\}$$

Prop: The addition of two polynomials f and g requires $(n+1)$ additions in R.

Theorem: The multiplication of polynomials of degree at most n in $R[x]$ needs
- $O(n^2)$ operations in R for naive algorithm
- $O(n^{1.59})$ operations in R for Karatsuba
- $O(n \log n \log\log n)$ (sometimes $O(n \log n)$) with the FAST FOURIER TRANSFORM (FFT).

Schönhage and Strassen (not in this course)

The naive algorithm: $F = f_0 + \dots + f_{n-1}x^{n-1}$ $\quad G = g_0 + \dots + g_{n-1}x^{n-1}$

$$H = FG = h_0 + \dots + h_{2n-2}x^{2n-2} = \sum_{i=0}^{2n-2} h_i x^i \quad \text{where} \quad h_i = \sum_{j+k=i} f_j g_k$$

Hence this demands $O(n^2)$ operations in R

Karatsuba Algorithm: We have already seen that in order to multiply $ax+b$ and $cx+d$ we can use 3 multiplications and 4 additions by doing $(ax+b)(cx+d) = acx^2 + ((a+b)(c+d) - ac - bd)x + bd$

Let F and G be two polynomials of degree at most $n-1$.
Let $k = \lceil \frac{n}{2} \rceil$. Let $F = F^{(0)} + F^{(1)}x^k$ and $G = G^{(0)} + G^{(1)}x^k$, where $F^{(0)}, F^{(1)}, G^{(0)}, G^{(1)}$ are polynomials of degree at most $k-1$.
Then we can write

$$H = F \cdot G = F^{(0)}G^{(0)} + \left(F^{(0)}G^{(1)} + F^{(1)}G^{(0)}\right)x^k + F^{(1)}G^{(1)}x^{2k}$$
$$= F^{(0)}G^{(0)} + \left((F^{(0)} + F^{(1)})(G^{(0)} + G^{(1)}) - F^{(0)}G^{(0)} - F^{(1)}G^{(1)}\right)x^k + F^{(1)}G^{(1)}x^{2k}$$

Karatsuba(F, G) — Input $F, G \in R[x]$, $\deg(F), \deg(G) \le n-1$; Output $FG \in R[x]$

If $n = 1$ then
  return FG
otherwise
  Build F_0, F_1, G_0, G_1 s.t. $F = F_0 + F_1x^k$, $G = G_0 + G_1x^k$
  $A = $ Karatsuba(F_0, G_0)
  $B = $ Karatsuba$(F_0 + F_1, G_0 + G_1)$
  $C = $ Karatsuba(F_1, G_1)
  return $Cx^n + (B - A - C)x^{n/2} + A$
end if

$\Rightarrow$ Karatsuba
$A_1 = $ Karatsuba(F_0, G_0)
$A_2 = $ Karatsuba(F_1, G_1)
$A_3 = F_0 + F_1$
$A_4 = G_0 + G_1$
$A_5 = $ Karatsuba(A_3, A_4)
$A_6 = A_5 - A_1$
$A_7 = A_6 - A_2$
return $A_1 + A_7x^k + A_2x^{2k}$

$3T(n/2) + 4n$

$\Rightarrow$ Complexity : $T(2n) = 3T(n) + 8n - 4$

$= 3T(n) + O(n)$

$T(n) = 3T(n/2) + O(n)$

Master Theorem: $a = 3$, $p = 2$

$\log_p a = \log_2 3$

Compare $n^{\log_2 3}$ with n

$n^{\log_2 3} > n \Rightarrow$ CASE 1 of MT $\rightarrow$ $T(n) = O(n^{\log_2 3})$

There is an improvement on the algorithm of Karatsuba. This is called Toom-Cook. For $n = r^k$, it decomposes the polynomials of degree up to n in r blocks of size n/r.

FAST FOURIER TRANSFORM (FFT)

The methods on which the FFT is based are the best we know so far for the multiplication of polynomials. For simplicity, we assume that we want multiply two polynomials $F, G \in R[x]$ with degree $< \frac{n}{2}$ (in other words $\deg(FG) < n$)

Def:
- An element $w \in R$ is an *n-th root of unity* if $w^n = 1$.
- An element $w \in R$ is a *primitive n-th root of unity* if $w^n = 1$ and $w^t \neq 1 \ \forall t < n$.
- An element $w \in R$ is a *principal n-th root of unity* if it is a primitive root of unity and $w^t - 1$ is not a zero divisor in R for $t \in \{1, \dots, n-1\}$ (i.e. $\alpha(w^t - 1) = 0 \Rightarrow \alpha = 0$).

Example let $R = \mathbb{Z}/25\mathbb{Z}$. The first power of 6 are $6, 11, 16, 21, 1$

$\Rightarrow$ 6 is a 5-th root of unity which is also primitive.

Note that $6^3 - 1 = 15$, which is a 0 divison

($5 \cdot 15 = 0$ in $\mathbb{Z}/25\mathbb{Z}$)

Hence 6 is not principal.

Examples: if R is a field, the principal and primitive roots of unity coincide. If w is a primitive n-th root of unity then w generates the group of n-th roots of unity.

If $R = \mathbb{C}$, the n-th root of unity are of the form $\exp\left(\frac{2qi\pi}{n}\right)$ for $\gcd(q, n) = 1$.

The idea of the algorithm is to evaluate the polynomials in precise points and then multiply the evaluations in order to reconstruct the coefficients of the product. If F and G coincide once evaluated in $1, w, \dots, w^{n-1}$, we will see that their difference is a multiple of $x^n - 1$.

Input of the algorithm: F, G polynomials, ω n-th root of unity

Output: $FG \bmod x^n - 1$ (FG if $\deg(FG) < n$)

1. Precompute $\omega^2, \omega^3, \dots, \omega^{n-1}$
2. Compute: $Ev(F) = (F(\omega^0), F(\omega), \dots, F(\omega^{n-1}))$

   $Ev(G) = (G(\omega^0), G(\omega), \dots, G(\omega^{n-1}))$
3. Product: $(Ev(F), Ev(G)) \mapsto Ev(FG) = (FG(\omega^0), \dots, FG(\omega^{n-1}))$
4. Interpolation: $Ev(FG) \longmapsto FG$

To understand the idea of the algorithm we need to prove some properties of the principal roots of unity.

Lemma: If ω is a primitive or principal n-th root of unity, then (TD)

1. ω^{-1} is also a primitive or principal n-th root of unity
2. If $n = pq$ then ω^p is a q-th root of unity of the same nature as ω
3. For $\ell \in \{1, \dots, n-1\}$ and ω principal, we have

$$\sum_{j=0}^{n-1} \omega^{\ell j} = 0$$

Proof: 1. First of all ω is invertible. Indeed $\omega^n = 1 \Rightarrow \omega^{n-1}$ is the inverse of ω. Hence, ω^{-1} is an n-th root of unity.

$\omega^n = 1$ multiplied by $\omega^{-n} \Rightarrow 1 = \omega^{n-n} = \omega^{-n}$

Finally, ω^{-1} is principal if ω is principal: when $\omega^t - 1$ is not a zero divisor, $(\omega^t - 1)\cdot\omega^{-t} \left(= 1 - \omega^{-t}\right)$ also is not a zero divisor. Same for the primitive.

2. $\omega^m = 1 \Rightarrow (\omega^p)^q = 1$.
3. Observe that

$$(1 - \omega^\ell) \sum_{j=0}^{n-1} \omega^{\ell j} = 1 - \omega^{\ell n} = 1 - (\omega^n)^\ell = 0.$$

Since $1 - \omega^\ell$ is not a zero divisor, $\sum_{j=0}^{n-1} \omega^{\ell j} = 0$. ∎

Remark: In the definition of principal or primitive n-th root of unity, it is sufficient to consider $t \in \{1, \dots, n-1\}$ s.t. $t \mid n$. Indeed if $t \nmid n$, $\gcd(t,n) \mid n$, hence, by Bezout theorem, $\exists$ integers $p, q \in \mathbb{N}$ s.t. $\underbrace{\gcd(t,n)}_{g} = tp + nq$

Hence $\alpha(\omega^t - 1) = 0 \Rightarrow$

$$0 = \alpha(\omega^t - 1)(1 + \omega^t + \dots + \omega^{t(p-1)}) =$$

$$= \alpha(\omega^{tp} - 1) = \alpha(\omega^{g - nq} - 1) = \alpha(\omega^g - 1) \Rightarrow \alpha = 0$$

since $g \mid n$ and $g < n$.

Def: The map $DFT_\omega : \begin{cases} R[X] \longrightarrow R^n \\ F \longmapsto (F(1), F(\omega), \dots, F(\omega^{n-1})) \end{cases}$

where ω is a principal n-th root of unity is called DISCRETE FOURIER TRANSFORM.

The fast computation of $DFT(F)$ is done with a divide and conquer type algorithm.

Decompose the polynomial $F = \sum_{i=0}^{n-1} f_i x^i$ in

Assume n = power of 2

$$F_{even} = \sum_{k=0}^{\frac{n}{2}} f_{2k} x^k \quad \text{and} \quad F_{odd} = \sum_{k=0}^{\frac{n}{2}-1} f_{2k+1} x^k$$

and observe that

$$\boxed{F = F_{even}(x^2) + x F_{odd}(x^2)} \quad (*)$$

Hence we will apply the algorithm on F_{even} and F_{odd} in order to compute the evaluation of F in w^i $0 \le i \le n-1$

Input: $F = f_0 + f_1 x + \cdots + f_{n-1} x^{n-1}$

where n is a power of 2

w principal (or primitive) n-th root of unity

Output: $F(1), F(w), \ldots, F(w^{n-1})$

DFT(F, ω, n)

if n = 1

return f_0

else

compute F_{even} and F_{odd} — DFT(F_{even}, ω^2), DFT(F_{odd}, ω^2)

compute recursively $\begin{cases} F_{even}(\omega^0), F_{even}(\omega^2), F_{even}(\omega^4), \dots, F_{even}(\omega^{n-2}) \\ F_{odd}(\omega^0), F_{odd}(\omega^2), F_{odd}(\omega^{n-2}) \end{cases}$

for $k \in \{0, \dots, n-1\}$ do

$F(\omega^k) = F_{even}(\omega^{2k}) + \omega^k F_{odd}(\omega^{2k})$

end for

return $F(\omega^0), \dots, F(\omega^{n-1})$

end if

Proposition: The complexity of DFT is $O(n \log n)$.

Proof: Let $C(n)$ be the computational cost, then

$$C(n) = 2C\left(\frac{n}{2}\right) + O(n)$$

Master theorem $\Rightarrow a = 2, p = 2$

Compare n with $n^{\log_p a} = n$

$\Rightarrow$ Case 2: $O(n \log n)$ ∎

---

* We want to compute the evaluations of F in ω^i $\forall\ 0 \le i \le n-1$

Observe that $F(\omega^i) = F_{even}(\omega^{2i}) + \omega^i F_{odd}(\omega^{2i})$

Now, ω^2 is a primitive $\frac{n}{2}$-th root of unity and

$$DFT_{\omega^2}(F_{even}) = (F_{even}(\omega^{2i}))_{i \in \{0, \dots, \frac{n}{2}\}} \quad \text{and} \quad DFT_{\omega^2}(F_{odd}) = (F_{odd}(\omega^{2i}))_{i \in \{0, \dots, \frac{n}{2}\}}$$

Assume we computed those values. Then $\forall\ \ell \in \{0, \dots, \frac{n}{2} - 1\}$ (*) allows to compute $F(\omega^\ell)$ since $\omega^{\frac{n}{2}} = -1$: $F(\omega^{\ell + \frac{n}{2}}) = F_{even}(\omega^{2\ell}) - \omega^\ell F_{odd}(\omega^{2\ell})$.

$\Rightarrow$ $F(\omega^\ell) = F_{even}(\omega^{2\ell}) + \omega^\ell F_{odd}(\omega^{2\ell})$

$F(\omega^{\ell + \frac{n}{2}}) = F_{even}(\omega^{2\ell}) - \omega^\ell F_{odd}(\omega^{2\ell})$.

Example: Let α be a 4-th root of unity in $\mathbb{C}$

$\alpha^4 = 1, \quad \alpha^2 = -1, \quad \alpha = i$

Compute the discrete fourier Transform of

$F = x^3 + 2x^2 + 3x + 4 \in \mathbb{C}[x]$

$DFT(F, 4, i)$

$F_{even} = 4 + 2x$

$F_{odd} = 3 + x$

$DFT(F_{even}, 2, \underset{\alpha^2}{-1})$ and $DFT(F_{odd}, 2, \underset{\alpha^2}{-1})$

$F_{even/even} = 4$ and $F_{odd/even} = 3$

$F_{even/odd} = 2$ $\quad$ $F_{odd/odd} = 1$

$DFT(F_{even/even}, 1, \underset{\alpha^4}{1}) = 4$ $\quad$ $DFT(F_{odd/even}, 1, 1) = 3$

$DFT(F_{even/odd}, 1, 1) = 2$ $\quad$ $DFT(F_{odd/odd}, 1, 1) = 1$

$F_{even}(-1) = F_{even}(\alpha^2) = F_{even/even}(\alpha^4) + \alpha^2 F_{even/odd}(\alpha^4)$

$= 4 - 2 = 2$

$F_{odd}(-1) = F_{odd}(\alpha^2) = F_{odd/even}(\alpha^4) + \alpha^2 F_{odd/odd}(\alpha^4)$

$= 3 - 1 = 2$

- $F(1) = F_{even}(1) + F_{odd}(1)$

  $= F_{even/even}(1) + F_{even/odd}(1) + F_{odd/even}(1) + F_{odd/odd}(1)$

  $= 4 + 2 + 3 + 1 = 10$

- $F(\alpha) = F_{even}(\alpha^2) + \alpha F_{odd}(\alpha^2)$

  $= 2 + i2$

- $F(\alpha^2) = F_{even}(\alpha^4) + \alpha^2 F_{odd}(\alpha^4)$

  $= 6 - 4 = 2$

- $F(\alpha^3) = F(\alpha^{1+\frac{m}{2}}) = F_{even}(\alpha^2) - \alpha F_{odd}(\alpha^2)$

  $= 2 - i2$

check: $F(i) = F(\alpha) = i^3 + 2i^2 + 3i + 4 = -i - 2 + 3i + 4 = 2i + 2$ ✓

Now we computed the evaluation of F in the primitive n-th roots of unity powers. We need to obtain the polynomial $F \cdot G$ with the interpolation.

Note that the map $F \longrightarrow Ev(F)$ is linear and its matrix representation (if F has $\deg \leq n-1$) is

$$\begin{pmatrix} F(1) \\ F(\omega) \\ \vdots \\ F(\omega^{n-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \cdots & (\omega^2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix}}_{V_\omega} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Vandermonde matrix

Proposition: Let ω be a primitive n-th root of unity.

Then $V_{\omega^{-1}} \cdot V_\omega = n I_n$

Proof: ω is invertible hence $V_{\omega^{-1}}$ is well-defined.

The entry (i,j) of $V_{\omega^{-1}} \cdot V_\omega$ is

$$\sum_{k=0}^{n-1} \omega^{-(i-1)k} \omega^{(j-1)k} = \sum_{k=0}^{n-1} \omega^{(j-i)k}$$

For $0 < j-1 < n$, this sum is 0.

By the structure of the matrix also for $0 < i-j < n$ is 0.

Hence the coefficients outside the diagonal are all 0. On the diagonal, the sum has n terms all equal to 1, hence the result follows. ∎

Consequence: If we want to reconstruct F from the values of the evaluation, it is sufficient to apply the DFT with ω^{-1} on the polynomial $\overline{F} = \sum F(\omega^k) x^k$ and divide by n.

Conclusion : Fast Fourier transform

FFT(F, G, W)
R_F = DFT(F, n, w) list of evaluations of F in $w^0, \dots w^{n-1}$
R_G = DFT(G, n, w) list of evaluations of G in $w^0, \dots w^{n-1}$
For i in {0, —, n-1} do
$R[i] = R_F[i] R_G[i]$
end for
Return $\frac{1}{n}$ DFT(R, w^{-1}, n) $R = F \cdot G$

Proposition : FFT computes F·G in $O(n \log n)$ operations in R.

Question : Do principal nth root of unity always exist?

Remark
Let n be a power of 2 and assume that 2 is not invertible in R. Let w be a primitive n-th root of unity. Then there exists a t s.t. $w^t - 1 = -2$, hence $w^t - 1$ is a zero divison (in a finite ring)

$n = 2^m$
$w^{2^m} - 1 = 0$, moreover, w is root of $x^{2^m} - 1$

$x^{2^m} - 1 = (x^{2^{m-1}} - 1)(x^{2^{m-1}} + 1)$

w cannot be a root of $x^{2^{m-1}} - 1$ because it is primitive

$\Rightarrow w^{2^{m-1}} = -1$

$\Rightarrow$ For $t = 2^{m-1}$ we have that $w^t - 1 = w^{2^{m-1}} - 1 = -2$.

Nb: Invertible elements in a finite ring are zero divisors
Indeed let $a \in R$, non invertible, let $R = \{d_1, \dots, d_n\}$
Then $ad_1, \dots, ad_n \in R$ and they cannot be 1
So $\Rightarrow$ at least two are equal : $ad_i = ad_j \Leftrightarrow a(d_i - d_j) = 0$
$\rightarrow$ a is a zero divison.