Fast-Fourier Transform

and Applications of Generating Functions

Chengyuan Luo
SJTU-Paris Elite Institute of Technology
December 3, 2022

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Table of Contents

Complex Numbers

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Definition

A number of the form $a + ib$ $(a, b \in \mathbb{R}^2)$ is called a complex number, where i is the imaginary unit with $i^2 = -1$.

- Addition of complex numbers:

$$(a + ib) + (c + id) = (a + c) + i(b + d)$$

- Complex addition satisfies closure, commutativity, and associativity. There exists an additive identity 0 $(\forall x \in \mathbb{C},\ x + 0 = x)$, and every complex number has an additive inverse $(\forall x \in \mathbb{C},\ \exists y \in \mathbb{C},\ x + y = y + x = 0)$.

Complex Numbers

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Definition

A number of the form $a + ib$ $(a, b \in \mathbb{R}^2)$ is called a complex number, where i is the imaginary unit with $i^2 = -1$.

- Multiplication of complex numbers:

$$(a + ib)(c + id) = ac - bd + i(ad + bc)$$

- Complex multiplication satisfies closure, commutativity, associativity, and distributivity. There exists a multiplicative identity 1 ($\forall x \in \mathbb{C},\ 1x = x1 = x$), and every non-zero complex number has a multiplicative inverse ($\forall x \in \mathbb{C}^*,\ \exists y \in \mathbb{C},\ xy = yx = 1$).

Complex Numbers

- Let $z = a + ib$.

Modulus of a Complex Number

$$|z| = \sqrt{a^2 + b^2}$$

- If a, b are not both zero,

$$z = \frac{a}{\sqrt{a^2 + b^2}} + i\frac{b}{\sqrt{a^2 + b^2}}$$

- We can find θ such that $\cos(\theta) = \frac{a}{\sqrt{a^2+b^2}}$ and $\sin(\theta) = \frac{b}{\sqrt{a^2+b^2}}$.
- Therefore, we can write $z = a + ib$ as $|z|\,(\cos(\theta) + i\sin(\theta))$. This is the polar coordinate representation of a complex number.

Complex Numbers

- Therefore, we can write $z = a + ib$ as $|z|\left(\cos(\theta) + i\sin(\theta)\right)$. This is the polar coordinate representation of a complex number.

Euler's Formula

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

- So we can express z in the form $|z|e^{i\theta}$.

Roots of Unity

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

n-th Roots of Unity

The complex roots z of the equation

$$z^n = 1 \ (n \in \mathbb{N}^*)$$

are called the n-th roots of unity.

- It can be verified that there are exactly n n-th roots of unity:
$$e^{i\frac{2\pi}{n}k} \ (k = 0, 1, \ldots, n-1)$$
- Because $$\left(e^{i\frac{2\pi}{n}k}\right)^n = e^{i2\pi k} = \cos(2\pi k) + i\sin(2\pi k) = 1$$
- Let $\omega_n = e^{i\frac{2\pi}{n}}$. Properties:
  1. $\omega_n^k = \omega_{2n}^{2k}$
  2. $\omega_n^k = \omega_n^{k \bmod n}$

Fourier Transform

- From Chapter 4 of Signaux (Signal Theory) in the first semester of sophomore year:

Fourier Transform

For a function f with period T, find the Fourier series:

$$\tilde{C}_n = \frac{2}{T} \int_{t_0}^{t_0+T} f(t) \exp\left(-j\frac{2\pi}{T}nt\right) \mathrm{d}t.$$

We can derive:

$$f(t) = \mathrm{Re}\left(\sum_{n=0}^{+\infty} \tilde{C}_n \exp\left(j\frac{2\pi}{T}nt\right)\right)$$

Discrete Fourier Transform

- Discrete Fourier Transform: Discrete Fourier Transform (Transformation de Fourier discrète)

Discrete Fourier Transform

For a sequence $(s_n)_{n\in[\![0,N-1]\!]}$, its discrete Fourier transform sequence (S_n) satisfies: $\forall\ n \in [\![0, N-1]\!]$,

$$S_n = \sum_{k=0}^{N-1} s_k \exp\left(-i\frac{2\pi}{N}kn\right)$$

Discrete Fourier Transform

Continuous Fourier Transform	Discrete Fourier Transform
$\tilde{C}_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\exp\left(-jn\omega t\right)\mathrm{d}t$	$S_n = \sum_{k=0}^{N-1} s_k \exp\left(-i\frac{2\pi}{N}kn\right)$
T	N
n	n
t	k

Discrete Fourier Transform

Inverse Discrete Fourier Transform

Inverse transformation (Inverse DFT, or IDFT)

$$s_k = \frac{1}{N} \sum_{n=0}^{N-1} S_n \exp\left(i \frac{2\pi}{N} kn\right)$$

Discrete Fourier Transform

Continuous Fourier Transform	Discrete Fourier Transform
$\tilde{C_n} = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\exp\left(-jn\omega t\right)\mathrm{d}t$	$S_n = \sum_{k=0}^{N-1} s_k \exp\left(-i\frac{2\pi}{N}kn\right)$
T	N
n	n
t	k
$f(t) = \mathrm{Re}\left(\sum_{n=0}^{+\infty}\tilde{C}_n\exp\left(j\frac{2\pi}{T}nt\right)\right)$	$s_k = \frac{1}{N}\sum_{n=0}^{N-1} S_n \exp\left(i\frac{2\pi}{N}kn\right)$
T	N
t	k
n	n

Discrete Fourier Transform

- Proving the inverse transformation:
- Let $\omega = \exp\left(i\frac{2\pi}{N}\right)$. $\forall\ k \in \mathbb{Z}$,

$$\sum_{n=0}^{N-1} \omega^{kn} = \sum_{n=0}^{N-1} (\omega_k)^n = \begin{cases} N, & N \mid k, \\ \frac{1-\omega^{kN}}{1-\omega^k} = \frac{1-\exp\left(i\frac{2\pi}{N}kN\right)}{1-\omega^k} = 0, & N \nmid k \end{cases}$$

- $\forall\ 0 \le k < N$,

$$\begin{aligned} s_k &= \frac{1}{N}\sum_{n=0}^{N-1} \omega^{kn} S_n = \frac{1}{N}\sum_{n=0}^{N-1} \omega^{kn} \sum_{r=0}^{N-1} \omega^{-rn} s_r \\ &= \frac{1}{N}\sum_{r=0}^{N-1} s_r \sum_{n=0}^{N-1} \omega^{(k-r)n} = \frac{1}{N}\sum_{r=0}^{N-1} s_r [k=r] N = s_k \end{aligned}$$

Discrete Fourier Transform

- $\forall\ 0 \le n < N$,

$$S_n = \sum_{k=0}^{N-1} \omega^{-kn} s_k = \sum_{k=0}^{N-1} \omega^{-kn} \frac{1}{N} \sum_{r=0}^{N-1} \omega^{kr} S_r$$
$$= \sum_{r=0}^{N-1} S_r \frac{1}{N} \sum_{k=0}^{N-1} \omega^{(r-n)k} = \sum_{r=0}^{N-1} S_r \frac{1}{N} [r = n] N = S_n$$

$$\mathrm{IDFT} \circ \mathrm{DFT} = \mathrm{DFT} \circ \mathrm{IDFT} = \mathrm{id}_{\mathbb{C}^{[\![0, N-1]\!]}}.$$

Fast Fourier Transform

- If we perform DFT according to the definition, calculating $S_n = \sum_{k=0}^{N-1} \omega^{-kn} s_k$ for each $n \in [\![0, N-1]\!]$, we need to first enumerate $n \in [\![0, N-1]\!]$ and then enumerate $k \in [\![0, N-1]\!]$, resulting in a time complexity of $O(N^2)$. The same applies to IDFT.
- There exists a faster method, the Fast Fourier Transform (FFT).
- One common implementation is the Cooley-Tukey algorithm. Using a divide-and-conquer approach, it improves the time complexity to $O(N \log N)$ (with $\log$ base 2).

Cooley-Tukey

- Consider the case $N = 4$ with s_0, s_1, s_2, s_3 $(\omega = \exp\left(i\frac{2\pi}{4}\right))$.
- We split (s_0, s_1, s_2, s_3) into two sequences: $(t_0 = s_0, t_1 = s_2)$ and $(t'_0 = s_1, t'_1 = s_3)$, and perform Fourier transforms on each sequence separately, obtaining $T_0 = s_0 + s_2$, $T_1 = s_0 + \omega^{-2}s_2$, $T'_0 = s_1 + s_3$, $T'_1 = s_1 + \omega^{-2}s_3$.
- Now we want to compute the Fast Fourier Transform of (s_0, s_1, s_2, s_3):

$$\begin{cases} S_0 = s_0 + s_1 + s_2 + s_3 = T_0 + T'_0, \\ S_2 = s_0 + \omega^{-2}s_1 + \omega^{-4}s_2 + \omega^{-6}s_3 = T_0 + \omega^{-2}T'_0 = T_0 - T'_0 \\ S_1 = s_0 + \omega^{-1}s_1 + \omega^{-2}s_2 + \omega^{-3}s_3 = T_1 + \omega^{-1}T'_1, \\ S_3 = s_0 + \omega^{-3}s_1 + \omega^{-6}s_3 + \omega^{-9}s_3 = T_1 + \omega^{-3}T'_1 = T_1 - \omega^{-1}T'_1. \end{cases}$$

Cooley-Tukey

- Extending to the case $N = 8$ ($\omega = \exp\left(i\frac{2\pi}{8}\right)$):

$$\begin{cases} S_0 = (s_0 + s_1 + s_2 + s_3) + (s_4 + s_5 + s_6 + s_7) = T_0 + T_0', \\ S_4 = (s_0 + s_2 + s_4 + s_6) + \omega^{-4}(s_1 + s_3 + s_5 + s_7) = T_0 + \omega^{-4}T_0' = T_0 - T_0', \\ S_1 = (s_0 + \omega^{-2}s_2 + \omega^{-4}s_4 + \omega^{-6}s_6) + \omega^{-1}(s_1 + \omega^{-2}s_3 + \omega^{-4}s_5 + \omega^{-6}s_7) = T_1 + \omega^{-1}T_1', \\ S_5 = (s_0 + \omega^{-2}s_2 + \omega^{-4}s_4 + \omega^{-6}s_6) + \omega^{-5}(s_1 + \omega^{-2}s_3 + \omega^{-4}s_5 + \omega^{-6}s_7) = T_1 - \omega^{-1}T_1', \\ S_2 = (s_0 + \omega^{-4}s_2 + s_4 + \omega^{-4}s_6) + \omega^{-2}(s_1 + \omega^{-4}s_3 + s_5 + \omega^{-4}s_7) = T_2 + \omega^{-2}T_2', \\ S_6 = (s_0 + \omega^{-4}s_2 + s_4 + \omega^{-4}s_6) + \omega^{-6}(s_1 + \omega^{-4}s_3 + s_5 + \omega^{-4}s_7) = T_2 - \omega^{-2}T_2', \\ S_3 = (s_0 + \omega^{-6}s_2 + \omega^{-4}s_4 + \omega^{-2}s_6) + \omega^{-3}(s_1 + \omega^{-6}s_3 + \omega^{-4}s_5 + \omega^{-2}s_7) = T_3 + \omega^{-3}T_3', \\ S_7 = (s_0 + \omega^{-6}s_2 + \omega^{-4}s_4 + \omega^{-2}s_6) + \omega^{-7}(s_1 + \omega^{-6}s_3 + \omega^{-4}s_5 + \omega^{-2}s_7) = T_3 - \omega^{-3}T_3'. \end{cases}$$

Cooley-Tukey

- Assume $N = 2^r$. Let $M = \frac{N}{2} = 2^{r-1}$.
- We have already performed Fourier transforms on $(s_0, s_2, \ldots, s_{N-2})$ and $(s_1, s_3, \ldots, s_{N-1})$ separately, obtaining $(T_0, T_1, \ldots, T_{M-1})$ and $(T'_0, T'_1, \ldots, T'_{M-1})$.

$$T_k = \sum_{n=0}^{M-1} s_{2n} \exp\left(-i\frac{2\pi}{M}kn\right)$$

$$T'_k = \sum_{n=0}^{M-1} s_{2n+1} \exp\left(-i\frac{2\pi}{M}kn\right)$$

Cooley-Tukey

$$
\begin{aligned}
S_k &= \sum_{n=0}^{N-1} s_n \exp\left(-i\frac{2\pi}{N}kn\right) \\
&= \sum_{n=0}^{M-1} s_{2n} \exp\left(-i\frac{2\pi}{N}k\cdot 2n\right) + \sum_{n=0}^{M-1} s_{2n+1} \exp\left(-i\frac{2\pi}{N}k\cdot(2n+1)\right) \\
&= \sum_{n=0}^{M-1} s_{2n} \exp\left(-i\frac{2\pi}{M}kn\right) + \exp\left(-i\frac{2\pi}{N}k\right) \sum_{n=0}^{M-1} s_{2n+1} \exp\left(-i\frac{2\pi}{M}kn\right) \\
&= T_k + T'_k \exp\left(-i2\pi\frac{k}{N}\right)
\end{aligned}
$$

- Or:

$$
S_k = \begin{cases} T_k + T'_k \exp\left(-i2\pi\frac{k}{N}\right), & k < M, \\ T_{k-M} - T'_{k-M} \exp\left(-i2\pi\frac{k-M}{N}\right), & k \geq M. \end{cases}
$$

Butterfly Transformation

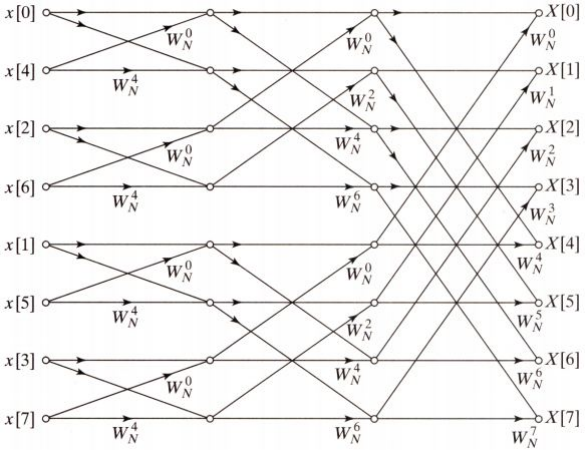

图: Butterfly Transformation[1]

Pseudocode

func dft($s_0, s_1, \ldots, s_{N-1}$):
    Result: $(S_0, S_1, \ldots, S_{N-1})$
    if $N = 1$ **then**
        return (s_0)
    else
        $M \leftarrow \frac{N}{2}$
        $(T_0, T_1, \ldots, T_{M-1}) \leftarrow$ dft$(s_0, s_2, \ldots, s_{N-2})$
        $(T'_0, T'_1, \ldots, T'_{M-1}) \leftarrow$ dft$(s_1, s_3, \ldots, s_{N-1})$
        for $k \leftarrow 0$ **to** $M - 1$ **do**
            $S_k \leftarrow T_k + T_{k+M} \exp\left(-i2\pi \frac{k}{N}\right)$
            $S_{k+M} \leftarrow T_k - T_{k+M} \exp\left(-i2\pi \frac{k}{N}\right)$
        end
        return $(S_0, S_1, \ldots, S_{N-1})$
    end

Cooley-Tukey

- For the inverse DFT: we only need to change $\exp\left(-i2\pi\frac{k}{N}\right)$ to $\exp\left(i2\pi\frac{k}{N}\right)$.

func `idft`$(s_0, s_1, \ldots, s_{N-1})$**:**
  Result: $(S_0, S_1, \ldots, S_{N-1})$
  if $N = 1$ **then**
    return (s_0)
  else
    $M \leftarrow \frac{N}{2}$
    $(T_0, T_1, \ldots, T_{M-1}) \leftarrow$ `idft`$(s_0, s_2, \ldots, s_{N-2})$
    $(T'_0, T'_1, \ldots, T'_{M-1}) \leftarrow$ `idft`$(s_1, s_3, \ldots, s_{N-1})$
    for $k \leftarrow 0$ **to** $M - 1$ **do**
      $S_k \leftarrow T_k + T_{k+M} \exp\left(+i2\pi\frac{k}{N}\right)$
      $S_{k+M} \leftarrow T_k - T_{k+M} \exp\left(+i2\pi\frac{k}{N}\right)$
    end
    return $(S_0, S_1, \ldots, S_{N-1})$
  end

Cooley-Tukey

- Analyzing its time complexity.
- Let $T(n)$ represent the number of operations for performing FFT on a sequence of length n.
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, where $O(n)$ means the number of operations can be approximated as a constant multiplied by n.
- From the Master Theorem[2], we get $T(n) = O(n \log n)$, which is significantly more efficient than $O(n^2)$.
- This is FFT with base 2 (splitting into two halves each time), but other bases or mixed bases can also be used.

Polynomial Multiplication

- Let's observe the Fourier transform formula again:

$$S_n = \sum_{k=0}^{N-1} s_k \exp\left(-i\frac{2\pi}{N}kn\right)$$
$$= \sum_{k=0}^{N-1} s_k \left(\exp\left(-i\frac{2\pi}{N}n\right)\right)^k$$

- Define

$$F(x) = \sum_{i=0}^{N-1} s_i x^i$$

- Then

$$S_n = F\left(\exp\left(-i\frac{2\pi}{N}n\right)\right)$$

Polynomial Multiplication

- Therefore, the discrete Fourier transform can obtain the values of a function at points $x = \exp\left(-i\frac{2\pi}{N}n\right)\ (0 \le n < N)$.
- The inverse discrete Fourier transform can recover the original function from its values at points $x = \exp\left(-i\frac{2\pi}{N}n\right)$.
- An N-th degree polynomial function can be uniquely determined by knowing its values at N+1 points.
- For an M-th degree function $F(x)$ and an R-th degree function $G(x)$, their product $(FG)(x) = F(x)G(x)$ is an $(M+R)$-th degree function.
- We choose $N = 2^{\lceil \log_2 (M+R+1) \rceil}$. We perform discrete Fourier transforms to obtain the values of F and G at points $x = \exp\left(-i\frac{2\pi}{N}n\right)\ (0 \le n < N)$.
- $\forall x \in \mathbb{C}$, $(FG)(x) = F(x)G(x)$. Therefore, we obtain the values of FG at points $x = \exp\left(-i\frac{2\pi}{N}n\right)$. After inverse transformation, we get the polynomial FG.

Polynomial Multiplication

- This is equivalent to quickly calculating the convolution of two sequences:

$$c_n = \sum_{k=0}^{n} a_k b_{n-k}$$

- The time complexity is $O(n \log n)$.

Polynomial Multiplication

- In practical numerical calculations, since both the real and imaginary parts of the roots of unity are decimal numbers, errors are unavoidable.
- In programming contests, we generally use $p = 998244353$ as the modulus:
  - 998244353 is a prime number, and $\mathbb{Z}/p\mathbb{Z}$ is a field (we can define addition, subtraction, multiplication, and division; every non-zero x has a multiplicative inverse x^{-1} such that $x \cdot x^{-1} \equiv 1 \pmod{p}$, because according to Bézout's identity, the equation $x \cdot x^{-1} - p \cdot y = 1$ has a solution).
  - 998244353 has a primitive root 3 ($\forall r \in [\![1, p-2]\!]$, $3^r \not\equiv 1 \pmod{p}$). Therefore, 3 satisfies the properties of complex roots of unity. (Euler's theorem: $\forall (a, p) = 1$, $a^{\varphi(p)} \equiv 1 \pmod{p}$)
  - $998244353 = 7 \times 17 \times 2^{23} + 1$, and $2^{23} \mid \varphi(p) = p - 1$. For any $N = 2^r (r \leq 23)$, we can compute $3^{\frac{p-1}{N}}$ as a complex root of unity.
- Fast Number Theoretic Transform (Number-theoretic transform).
- There are also fast number theoretic transform algorithms with other moduli, such as the MTT algorithm named after Mao Xiao[3].

Polynomial Inversion

- For $N \in \mathbb{N}^*$ and a polynomial $P(x)$ $(p_0 \neq 0)$, find a polynomial $Q(x)$ such that $P(x)Q(x) \equiv 1 \pmod{x^N}$.
- Let $p_0, p_1, \ldots, p_{N-1}$ be the coefficients of $x^0, x^1, \ldots, x^{N-1}$ in $P(x)$. Similarly for $q_0, q_1, \ldots, q_{N-1}$.

First Method

- $q_0 = \frac{1}{p_0}$;
- Recursively compute q_k $(1 \leq k < N)$:

$$\sum_{i=0}^{k} p_i q_{k-i} = p_0 q_k + \sum_{i=1}^{k} p_i q_{k-i} = 0$$

$$q_k = -\left(\sum_{i=1}^{k} p_i q_{k-i}\right) / p_0$$

- The algorithm complexity is $O(N^2)$.

Polynomial Inversion: Second Method

- The above equation is in the form of a convolution.
- We want to optimize using FFT, but we need to find the values of $q_0, q_1, \ldots, q_{i-1}$ before we can find q_i.
- We use the idea of CDQ divide and conquer (Chen Danqi's divide and conquer[4]) to solve in batches and perform convolution.

Polynomial Inversion Using Divide and Conquer

- For example, we now need to compute $q_0, q_1, \ldots, q_n$.
- Define $t_k = \sum_{i=1}^{k} p_i q_{k-i}$. Then $q_k = -t_k/p_0$. So the key is to compute t_k.
- Assume we already have $q_0, q_1, \ldots, q_m$ (this step can be considered as recursive solving).
- First consider the contribution of $q_0, \ldots, q_m$ to t_k $(m+1 \le k \le n)$. That is, we compute $\sum_{1 \le i \le k, k-i \le m} p_i q_{k-i}$. This is obviously in the form of a convolution and can be accelerated using FFT.

Polynomial Inversion: Second Method

上海交通大學 Shanghai Jiao Tong University

CDQ Divide and Conquer

Suppose a sequence $(f_0, f_1, \ldots, f_n)$ has a recurrence relation.

func `solve`(l, r)**:**

- **Result:** Find $f_l, f_{l+1}, \ldots, f_r$
- **if** $l < r$ **then**
  - $mid \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$
  - `solve`(l, mid)
  - Calculate the influence of $f_l, \ldots, f_{mid}$ on $f_{mid+1}, \ldots, f_r$
  - `solve`$(mid + 1, r)$
- **else**
  - Find f_l
- **end**

Polynomial Inversion: Second Method

- Correctness: $\forall i$, before finding f_i, we have considered the influence of $f_0, f_1, \ldots, f_{i-1}$ on f_i.
- Time Efficiency:
  - The influence of $f_l, \ldots, f_{mid}$ on $f_{mid+1}, \ldots, f_r$ is a convolution, so it can be accelerated using FFT.
  - Time complexity $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$. According to the Master Theorem[2], we get $T(n) = O(n \log^2 n)$.

Polynomial Inversion: Third Method

Taylor Expansion

- If a function $f(x)$ is n-times differentiable at a point x_0 in its domain, then

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \dots \\ &\quad + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + o_{x \to x_0}\left((x - x_0)^n\right) \\ &= \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i + o_{x \to x_0}\left((x - x_0)^n\right) \end{aligned}$$

- This means that in the neighborhood of x_0, we can approximate $f(x)$ as a polynomial.
- It can be verified that the original function and its Taylor expansion have the same derivatives from order 0 to n.

Polynomial Inversion: Third Method

Newton's Method

- We want to find a zero of a function $f(x)$, i.e., $f(x_0) = 0$.
- We use an iterative method to continuously approximate the zero.
- Let x_1 be the current iterated point. We perform a first-order Taylor expansion of f at x_1.

$$f(x) = f(x_1) + f'(x_1)(x - x_1) + o_{x \to x_1}(x - x_0)$$

Solving

$$f(x_1) + f'(x_1)(x - x_1) = 0$$

$$x = x_1 - \frac{f(x_1)}{f'(x_1)}$$

- $x_1 \leftarrow x_1 - \frac{f(x_1)}{f'(x_1)}$ and iterate again.

Polynomial Inversion: Third Method

- Now we want to find $Q(x) \equiv (P(x))^{-1} \pmod{x^N}$ (we will omit (x) from now on).
- We define the function $F(A) = \frac{1}{A} - P$. Then $F(Q) \equiv 0 \pmod{x^N}$.
- Consider Newton's method: Suppose we have found $Q_0 \equiv P^{-1} \pmod{x^{\lceil N/2 \rceil}}$. Now we want to find $Q_1 \equiv P^{-1} \pmod{x^N}$, i.e., find the coefficients of P^{-1} from $\lceil N/2 \rceil$ to $N-1$.
- $Q_0 \equiv Q_1 \pmod{x^{\lceil N/2 \rceil}}$.

$$\begin{aligned} F(Q_1) &\equiv F(Q_0) + F'(Q_0)(Q_1 - Q_0) + \frac{F''(Q_0)}{2}(Q_1 - Q_0)^2 + \dots \\ &\equiv F(Q_0) + F'(Q_0)(Q_1 - Q_0) \equiv 0 \pmod{x^N} \\ Q_1 &\equiv Q_0 - \frac{F(Q_0)}{F'(Q_0)} \pmod{x^N} \end{aligned}$$

Polynomial Inversion: Third Method

- $F'(A) = \frac{\mathrm{d}F}{\mathrm{d}A}(A) = -\frac{1}{A^2}$

$$Q_1 \equiv Q_0 + Q_0^2\left(\frac{1}{Q_0} - P\right) \pmod{x^N}$$

$$Q_1 \equiv 2Q_0 - PQ_0^2 \pmod{x^N}$$

- We can verify: $PQ_1 \equiv P(2Q_0 - PQ_0)^2 \equiv PQ_0(2 - PQ_0) \pmod{x^N}$
- $PQ_0 \equiv 1 \pmod{x^{\lceil N/2 \rceil}}$. So there exists a polynomial R such that $PQ_0 \equiv 1 + x^{\lceil N/2 \rceil}R \equiv 1 \pmod{x^N}$.
- $PQ_1 \equiv \left(1 + x^{\lceil N/2 \rceil}R\right)\left(1 - x^{\lceil N/2 \rceil}R\right) \equiv 1 - \left(x^{\lceil N/2 \rceil}R\right)^2 \equiv 1 \pmod{x^N}$

Polynomial Inversion: Third Method

- Pseudocode:

  func `getInv(`P, N`)`**:**

  > **Result:** $Q \equiv P^{-1} \pmod{x^N}$
  > **if** $N = 1$ **then**
  > > **return** p_0^{-1}
  > **else**
  > > $M \leftarrow \lceil \frac{N}{2} \rceil$
  > > $Q_0 \leftarrow$ `getInv(`$P \bmod x^M, M$`)`
  > > **return** $Q_0(2 - PQ_0) \bmod x^N$
  > **end**

- Time complexity $T(N) = T\left(\frac{N}{2}\right) + O(N \log N)$. According to the Master Theorem[2], we get $T(N) = O(N \log N)$.

Polynomial Division

- Given an N-th degree polynomial A and an M-th degree polynomial B ($M < N$, $a_N, b_M \neq 0$), find polynomials Q and R such that $A = BQ + R$ and $\deg(R) < M$.
- We can conclude that Q is an $(N - M)$-th degree polynomial, with $q_{N-M} = \frac{a_N}{b_M}$, and then we can recursively derive $q_{N-M-1}, q_{N-M-2}, \ldots, q_0$. After finding Q, we can compute $R = A - BQ$.
- We observe that the coefficients are determined from highest to lowest degree. If we follow the idea of polynomial inversion, we need to reverse the coefficients of these two polynomials.
- $\bar{A}(x) = x^N A\left(\frac{1}{x}\right) = x^N \sum_{i=0}^{N} a_i x^{-i} = \sum_{i=0}^{N} a_i x^{N-i} = \sum_{j=0}^{N} a_{N-j} x^j$.
- Similarly, $\bar{B}(x) = x^M B\left(\frac{1}{x}\right)$, $\bar{Q}(x) = x^{N-M} Q\left(\frac{1}{x}\right)$, and $\bar{R}(x) = x^{M-1} Q\left(\frac{1}{x}\right)$.

Polynomial Division

$$
\begin{aligned}
A(x) &= B(x)Q(x) + R(x) \\
x^N A\left(\frac{1}{x}\right) &= x^M B\left(\frac{1}{x}\right) x^{N-M} Q\left(\frac{1}{x}\right) + x^{N-M+1} \cdot x^{M-1} R\left(\frac{1}{x}\right) \\
\bar{A}(x) &= \bar{B}(x)\bar{Q}(x) + x^{N-M+1}\bar{R}(x) \\
\bar{A}(x) &\equiv \bar{B}(x)\bar{Q}(x) \pmod{x^{N-M+1}} \\
\bar{Q}(x) &\equiv \bar{A}(x) \cdot (\bar{B}(x))^{-1} \pmod{x^{N-M+1}}
\end{aligned}
$$

- We can use polynomial inversion to find $\bar{Q}(x)$, then reverse the coefficients to get $Q(x)$. The time complexity is $O(N \log N)$.

Polynomial Fast Exponentiation

- Polynomial ln: For a polynomial $P(x)$ $(p_0 = 1)$, find $\ln P(x)$.
  - $(\ln P(x))' = \frac{P'(x)}{P(x)}$
  - $\ln P(x) = \int \frac{P'(x)}{P(x)} \mathrm{d}x$ $([x^0] \ln P(x) = 0)$
  - $\ln P(x) \equiv \int P'(x)(P(x))^{-1} \mathrm{d}x \pmod{x^N}$
- Polynomial exp: For a polynomial $P(x)$ $(p_0 = 0)$, find $\exp(P(x)) \pmod{x^N}$:
  - $Q(x) \equiv \exp(P(x)) \equiv \sum_{i=0}^{+\infty} \frac{P(x)^i}{i!} \equiv \sum_{i=0}^{N-1} \frac{P(x)^i}{i!} \pmod{x^N}$
  - Define $F(A) = \ln(A) - P$. So $F'(A) = \frac{1}{A}$.
  - Newton's method: $Q_1 \equiv Q_0 - Q_0 (\ln(Q_0) - P) \pmod{x^N}$.
- Polynomial fast exponentiation: Find $P(x)^k \pmod{x^N}$.
  - Define $P'(x) = \frac{P(x)}{p_0}$. Then $[x^0]P'(x) = 1$ and $P(x) = p_0 P'(x)$.
  - $P(x)^k = p_0^k P'(x)^k = p_0^k \exp(k \ln P'(x))$.

Polynomial Multi-point Evaluation

- For an N-th degree polynomial function $P(x)$.
- Given M points $(x_1, x_2, \ldots, x_M)$, find $P(x_1), P(x_2), \ldots, P(x_M)$.
- In fact, $P(x_k) = P(x) \bmod (x - x_k)$.
- Proof:
  - Let $Q(x), R$ satisfy $Q(x)(x - x_k) + R = P(x)$.
  - Substituting x_k, we get $R = Q(x_k)(x_k - x_k) + R = P(x_k)$.

Polynomial Multi-point Evaluation

- We can solve this using divide and conquer.
- First, we compute $\prod_{i=1}^{M}(x - x_i)$ using divide and conquer, and then compute $P(x) \bmod \left(\prod_{i=1}^{M}(x - x_i)\right)$.
- Then we perform top-down divide and conquer. For example, $\texttt{solve}(l, r)$ has $P(x) \bmod (\prod_{i=l}^{r}(x - x_i))$ (denoted as $P_{l,r}(x)$).
- Let $mid = \left\lfloor \frac{l+r}{2} \right\rfloor$.
- Compute $P_{l,mid} = P_{l,r}(x) \bmod \left(\prod_{i=l}^{mid}(x - x_i)\right)$, and continue the divide and conquer for (l, mid). Similarly for $(mid + 1, r)$.
- When $l = r$, we get the value of the polynomial function at $x = x_l$.

Polynomial Multi-point Evaluation

func `getProd`$(l,\ r)$**:**
  if $l < r$ **then**
    $mid \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$
    return $Prod_{l,r} \leftarrow$ `getProd`$(l,\ mid) \times$ `getProd`$(mid+1,\ r)$
  else
    return $Prod_{l,l} \leftarrow x - x_l$
  end
func `getValue`$(l,\ r,\ P)$**:**
  if $l < r$ **then**
    $mid \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$
    `getValue`$(l,\ mid,\ P \bmod Prod_{l,mid})$
    `getValue`$(mid+1,\ r,\ P \bmod Prod_{mid+1,r})$
  else
    $res_l \leftarrow P$
  end

Polynomial Multi-point Evaluation

- Complexity analysis:
- `getProd` and `getValue`: $T(M) = 2T\left(\frac{M}{2}\right) + O(M \log M) = O(M \log^2 M)$.
- The total time complexity is $O(N \log N + M \log^2 M)$.

Polynomial Fast Interpolation

- Given N+1 points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ (with distinct x-coordinates), determine an N-th degree polynomial function P such that $\forall 0 \leq i \leq n$, $P(x_i) = y_i$.

Lagrange Interpolation

- Define

$$Q_k(x) = \prod_{0 \leq i \leq N, i \neq k} (x - x_i)$$

- The unique N-th degree polynomial function P satisfies

$$P(x) = \sum_{k=0}^{N} \frac{y_k}{Q_k(x_k)} Q_k(x)$$

Polynomial Fast Interpolation

- First, we solve how to compute $Q_k(x_k)$.
- $Q_k(x) = \frac{\prod_{i=0}^{N}(x-x_i)}{x-x_k}$.
- When $x = x_k$, both the numerator and denominator of $Q_k(x)$ are 0.
- L'Hôpital's rule: $Q_k(x_k) = \frac{\left(\prod_{i=0}^{N}(x-x_i)\right)'(x_k)}{(x-x_k)'(x_k)} = \left(\prod_{i=0}^{N}(x-x_i)\right)'(x_k)$.
- $\left(\prod_{i=0}^{N}(x-x_i)\right)'$ does not depend on k, so we can perform multi-point evaluation in $O(N\log^2 N)$ time.

Polynomial Fast Interpolation

- Now the problem is how to quickly find $\sum_{k=0}^{N} y'_k Q_k(x)$.
- We can split $Q_k(x)$ into $Prod_{0,k-1}(x) = \prod_{i=0}^{k-1}(x - x_i)$ and $Prod_{k+1,N}(x) = \prod_{i=k+1}^{N}(x - x_i)$.
- When we divide and conquer to $[l, r]$, we can simultaneously find $\prod_{k=l}^{r}(x - x_k)$ and $\sum_{k=l}^{r} y'_k \prod_{l \le i \le r, i \ne k}(x - x_i)$.

Polynomial Fast Interpolation

- Merge the answers for $[l, mid]$ and $[mid+1, r]$:

$$\prod_{k=l}^{r}(x-x_k)=\prod_{k=l}^{mid}(x-x_k)\cdot\prod_{k=mid+1}^{r}(x-x_k)$$

$$\sum_{k=l}^{r}y'_k\prod_{l\le i\le r,i\neq k}(x-x_i)=\left(\sum_{k=l}^{mid}y'_k\prod_{l\le i\le mid,i\neq k}(x-x_i)\right)\prod_{i=mid+1}^{r}(x-x_i)$$
$$+\left(\sum_{k=mid+1}^{r}y'_k\prod_{mid+1\le i\le r,i\neq k}(x-x_i)\right)\prod_{i=l}^{mid}(x-x_i)$$

- The time complexity is also $O(N\log^2 N)$.

Luogu P5850 calc Enhanced Version

- Problem Link
- Given m and k.
- A sequence $(a_1, a_2, \ldots, a_n)$ of length n is valid if and only if:
  1. $a_1, a_2, \ldots, a_n$ are all integers in $[1, k]$;
  2. $a_1, a_2, \ldots, a_n$ are distinct.
- For $\forall 1 \le n \le m$, find

$$\sum_{(a_1,\ldots,a_n) \text{ valid}} \prod_{i=1}^{n} a_i \bmod 998244353$$

- $m \le 5 \times 10^5$, $1 \le m \le k \le 998244352$ (requires an $O(m \log m)$ algorithm).

Luogu P5850 calc Enhanced Version Solution[5]

- Since $a_1, \ldots, a_n$ are distinct, we can first compute the answer for strictly increasing $a_1, \ldots, a_n$, then multiply by $n!$.
- The answer for n is $[x^n] \prod_{i=1}^{k} (1+ix)$. So we just need $\prod_{i=1}^{k} (1+ix) \bmod x^{m+1}$.

$$\begin{aligned}\prod_{i=1}^{k}(1+ix) &= \exp\left(\sum_{i=1}^{k}\ln(1+ix)\right)\\ &= \exp\left(\sum_{i=1}^{k}\sum_{j=1}^{+\infty}\frac{(-1)^{j-1}i^j x^j}{j}\right)\\ &= \exp\left(\sum_{j=1}^{+\infty}x^j\cdot\frac{(-1)^{j-1}}{j}\sum_{i=1}^{k}i^j\right)\end{aligned}$$

Luogu P5850 calc Enhanced Version Solution[5]

- Now the bottleneck is to compute $\sum_{i=1}^{k} i^j \ (1 \le j \le m)$, which is a typical problem of summing powers of natural numbers.

$$\sum_{j=0}^{+\infty} x^j \cdot \frac{1}{j!} \sum_{i=1}^{k} i^j = \sum_{i=1}^{k} \sum_{j=0}^{+\infty} \frac{(ix)^j}{j!}$$

$$= \sum_{i=1}^{k} \exp(ix) = \frac{\exp((k+1)x) - \exp(x)}{\exp(x) - 1}$$

- Therefore

$$\sum_{i=1}^{k} i^j = j![x^j] \frac{\exp((k+1)x) - \exp(x)}{\exp(x) - 1} = j![x^j] \frac{\frac{\exp((k+1)x) - \exp(x)}{x}}{\frac{\exp(x) - 1}{x}}$$

We can use polynomial inversion.

The Meaning of Polynomial $\exp$

Problem

- N labeled balls are divided into any number of unlabeled boxes. Each box can hold at most M balls.
- Given $f_1, f_2, \ldots, f_M$
- Suppose a division has m boxes, and the number of balls in each box is $a_1, a_2, \ldots, a_m$, then the weight of this division is $\prod_{i=1}^{m} f_{a_i}$.
- Find the sum of weights of all divisions.

- Assume only $f_k \neq 0$ and all other f are 0, meaning each box can only have size k.
- Let the number of balls be n $(k \mid n)$, then the sum of weights of these schemes is $\frac{n!}{(k!)^{n/k}(n/k)!} f_k^{n/k}$, which is $n![x^n] \exp\left(\frac{f_k}{k!}x^k\right)$.

The Meaning of Polynomial exp

- Now add boxes of different sizes. Assume there are a_k boxes of size k.
- The answer is

$$\sum_{(a_1,\ldots,a_M),\sum_{i=1}^{M} ia_i=N} \binom{N}{a_1, 2a_2, \ldots, Ma_M} \prod_{i=1}^{M} (ia_i)![x^{ia_i}] \exp\left(\frac{f_i}{i!}x^i\right)$$

$$= N! \sum_{\prod_{i=1}^{M} ia_i=N} [x^{ia_i}] \exp\left(\frac{f_i}{i!}x^i\right)$$

$$= N![x^N] \prod_{i=1}^{M} \exp\left(\frac{f_i}{i!}x^i\right) = N![x^N] \exp\left(\sum_{i=1}^{M} \frac{f_i}{i!}x^i\right)$$

Counting Labeled Simple Undirected Connected Graphs

Problem

There are N labeled points, find the number of simple undirected connected graphs.

1. Simple: no self-loops (edges from a point to itself), no multiple edges (no two edges connecting exactly the same pair of points).
2. Undirected: edges have no specified direction.
3. Connected: for any pair of points in the graph, there are several edges connecting these two points.

- First, ignore the "connected" requirement. A simple undirected graph is just deciding whether there is an edge between any pair of points, so the number of simple undirected graphs with n points is $2^{\binom{N}{2}}$.
- A simple undirected graph can be split into several simple undirected connected graphs, and there are no edges between these connected graphs.

Counting Labeled Simple Undirected Connected Graphs

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

- So place N labeled points into several boxes, each box corresponding to a connected graph, and a box containing m points corresponds to the number of simple undirected connected graphs with m points. The sum of weights of all schemes is the number of simple undirected graphs.
- This corresponds to the meaning of polynomial $\exp$. The $\exp$ of the EGF of the number of simple undirected connected graphs is the EGF of the number of simple undirected graphs. Taking the $\ln$, the final answer is $N![x^N] \ln \left(\sum_{n \geq 0} \frac{2^{\binom{n}{2}}}{n!} x^n \right)$

Linear Constant Coefficient Homogeneous Recurrence

- $f_n = \sum_{i=1}^{p} a_i f_{n-i}$ $(n > p)$.
- Let

$$A = \begin{bmatrix} a_p & a_{p-1} & \dots & a_2 & a_1 \\ 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

- Then

$$\begin{bmatrix} a_{p+n} \\ a_{p+n-1} \\ \vdots \\ a_{n+1} \end{bmatrix} = A^n \begin{bmatrix} a_p \\ a_{p-1} \\ \vdots \\ a_1 \end{bmatrix}$$

Linear Constant Coefficient Homogeneous Recurrence

- We can compute A^n using matrix exponentiation. But this has a time complexity of $O(p^3 \log n)$, which cannot handle cases where p is large.

Cayley-Hamilton Theorem[6]

For an $n \times n$ square matrix A, let its characteristic polynomial be $p(\lambda) = \det(\lambda I_n - A)$, then $p(A) = O$.

- It can be verified that:

$$p(\lambda) = \lambda^p - \sum_{i=1}^{p} a_p \lambda^{p-i}$$

- To find f_n, we need to find A^{n-p}.
- Consider $x^{n-p} = p(x)Q(x) + R(x)$ (polynomial division). Then

$$A^{n-p} = p(A)Q(A) + R(A) = R(A)$$

Linear Constant Coefficient Homogeneous Recurrence

- $R(x)$ is a polynomial of degree $p-1$. Let $R(x) = \sum_{i=0}^{p-1} r_i x^i$.

$$\begin{bmatrix} f_n \\ \vdots \\ f_{n-p+1} \end{bmatrix} = A^{n-p} \begin{bmatrix} f_p \\ \vdots \\ f_1 \end{bmatrix} = \sum_{i=0}^{p-1} r_i A^i \begin{bmatrix} f_p \\ \vdots \\ f_1 \end{bmatrix} = \sum_{i=0}^{p-1} r_i \begin{bmatrix} f_{i+p} \\ \vdots \\ f_{i+1} \end{bmatrix}$$

- To find f_n, we only need to consider the first row of the matrix:

$$f_n = \sum_{i=0}^{p-1} r_i f_{i+p}$$

- Quickly find $x^{n-p} \bmod p(x)$: fast exponentiation.
  - Compute $x^k \bmod p(x)$: if k is even, then $x^k \equiv \left(x^{k/2}\right)^2 \pmod{p(x)}$, otherwise $x^k \equiv x^{k-1} x \pmod{p(x)}$.
  - Using polynomial division, the time complexity is $O(p \log p \log n)$.

Linear Constant Coefficient Homogeneous Recurrence

- Quickly find $f_{i+1}, \ldots, f_{i+p-1}$:
  - OGF of f:

$$F(x) = \sum_{k=1}^{+\infty} f_k x^k$$

$$\forall n \le p,\ f_n - \sum_{i=1}^{p} a_i f_{n-i} = 0$$

$$F(x)\left(1 - \sum_{i=1}^{p} a_i x^i\right) = \sum_{i=1}^{p} x^i \left(f_i - \sum_{j=1}^{i-1} a_j f_{i-j}\right)$$

  - Divide both sides by $1 - \sum_{i=1}^{p} a_i x^i$, using polynomial inversion, with time complexity $O(p \log p)$.
- The total time complexity is $O(p \log p \log n)$.

Finding Particular Solutions for a Class of Differential Equations

Problem

Find

$$\sum_{i=0}^{n} a_i y^{(i)} = e^{\alpha x} \left(\sum_{j=0}^{m} b_j x^j \right)$$

a particular solution of the form $e^{\alpha x}$ multiplied by a polynomial ($a_n \neq 0$).

- First consider the derivative of $y = e^{\alpha x} x^k$.
- Define

$$k^{\underline{r}} = \begin{cases} \prod_{i=0}^{r-1} (k-i) & (r > 0) \\ 1 & (r = 0) \end{cases}$$

called the r-th falling factorial of k.

Finding Particular Solutions for a Class of Differential Equations

- Observe

$$y' = e^{\alpha x}\left(\alpha x^k + k x^{k-1}\right),$$
$$y^{(2)} = e^{\alpha x}\left[\alpha^2 x^k + 2\alpha k x^{k-1} + k^{\underline{2}} x^{k-2}\right],$$
$$y^{(3)} = e^{\alpha x}\left[\alpha^3 x^k + 3\alpha^2 k x^{k-1} + 3\alpha k^{\underline{2}} x^{k-2} + k^{\underline{3}} x^{k-3}\right]$$

- We can find the pattern:

$$y^{(r)} = e^{\alpha x}\sum_{i=0}^{r}\binom{r}{i}\alpha^{r-i}k^{\underline{i}}x^{k-i}$$

- This can be proven by mathematical induction.

Finding Particular Solutions for a Class of Differential Equations

- Proof by induction.
- Clearly true when $r = 0$.
- Assume $y^{(r)} = e^{\alpha x} \sum_{i=0}^{r} \binom{r}{i} \alpha^{r-i} k^{\underline{i}} x^{k-i}$.

$$
\begin{aligned}
y^{(r+1)} &= e^{\alpha x} \cdot \alpha \sum_{i=0}^{r} \binom{r}{i} \alpha^{r-i} k^{\underline{i}} x^{k-i} + e^{\alpha x} \cdot \sum_{i=0}^{r} \binom{r}{i} \alpha^{r-i} k^{\underline{i}} \cdot (k-i) x^{k-i-1} \\
&= e^{\alpha x} \left[\sum_{i=0}^{r} \binom{r}{i} \alpha^{r+1-i} k^{\underline{i}} x^{k-i} + \sum_{i=1}^{r+1} \binom{r}{i-1} \alpha^{r+1-i} k^{\underline{i-1}} (k-(i-1)) x^{k-i} \right] \\
&= e^{\alpha x} \left[\sum_{i=0}^{r+1} \binom{r}{i} \alpha^{r+1-i} k^{\underline{i}} x^{k-i} + \sum_{i=1}^{r+1} \binom{r}{i-1} \alpha^{r+1-i} k^{\underline{i}} x^{k-i} \right] \\
&= e^{\alpha x} \sum_{i=0}^{r+1} \binom{r+1}{i} \alpha^{r+1-i} k^{\underline{i}} x^{k-i}.
\end{aligned}
$$

Finding Particular Solutions for a Class of Differential Equations

- In fact, this coefficient $\binom{r}{i}\alpha^{r-i}k^{\underline{i}}$ can also be derived using combinatorial meaning.
- $\left(e^{\alpha x}x^k\right)' = e^{\alpha x}\left(kx^{k-1} + \alpha x^k\right)$. So for $e^{\alpha x}x^k$, there are two possibilities after differentiation:
  1. The degree of x decreases by 1, i.e., $x^k \to x^{k-1}$, and the coefficient is multiplied by k;
  2. The degree of x remains unchanged, $x^k \to x^k$, and the coefficient is multiplied by α.
- After r differentiation operations, $x^k \to x^{k-i}$, which means there are i operations of type 1 and $r-i$ operations of type 2.
- For operations of type 1, the product of these coefficients is $k(k-1)\ldots(k-i+1) = k^{\underline{i}}$.
- For operations of type 2, the product of these coefficients is α^{r-i}.
- There are $\binom{r}{i}$ ways to choose i operations of type 1 from r operations.
- So the coefficient is $\binom{r}{i}\alpha^{r-i}k^{\underline{i}}$.

Finding Particular Solutions for a Class of Differential Equations

- Therefore
$$\sum_{i=0}^{n} a_i y^{(i)} = e^{\alpha x} \sum_{j=0}^{n} k^{\underline{j}} x^{k-j} \sum_{r=0}^{n} \binom{r}{j} \alpha^{r-j} a_r$$
- Define
$$C_j = \sum_{i=j}^{n} \binom{i}{j} \alpha^{i-j} a_i$$
- Therefore
$$\sum_{i=0}^{n} a_i y^{(i)} = e^{\alpha x} \sum_{j=0}^{k} k^{\underline{j}} C_j x^{k-j}$$
- Enumerate $l = k - j$:
$$\sum_{i=0}^{n} a_i y^{(i)} = e^{\alpha x} \sum_{l=0}^{k} k^{\underline{k-l}} C_{k-l} x^{l}$$

Finding Particular Solutions for a Class of Differential Equations

- We take the smallest p such that $C_p \neq 0$.
- Then when $y = e^{\alpha x}x^k$, the coefficients of terms from x^k to x^{k-p+1} in $\sum_{i=0}^{n} a_i y^{(i)}$ are all 0, and the coefficient of the x^{k-p} term is not 0.
- Therefore, to find the solution of this differential equation, the highest degree term should be $e^{\alpha x}x^{m+p}$.
- We determine from highest degree to lowest degree,

$$\sum_{i=0}^{n} a_i (e^{\alpha x}x^{m+p})^{(i)} = e^{\alpha x}\sum_{l=0}^{m}(m+p)^{\underline{m+p-l}}C_{m+p-l}x^l$$

- To make the coefficient of $e^{\alpha x}x^m$ in $\sum_{i=0}^{n} a_i y^{(i)}$ equal to b_m, the coefficient of $e^{\alpha x}x^{m+p}$ should be $\frac{b_m}{(m+p)^{\underline{p}}C_p}$.

Finding Particular Solutions for a Class of Differential Equations

$$\frac{b_m}{(m+p)^{\underline{p}} C_p} e^{\alpha x} x^{m+p}$$

- The subsequent b_i $(i < m)$ should subtract the influence of this term, i.e.,

$$b_i \leftarrow b_i - \frac{b_m}{(m+p)^{\underline{p}} C_p} C_{m+p-i} (m+p)^{\underline{m+p-i}}$$

- Continue solving for the new differential equation.
- Let $z_p, \ldots, z_{m+p}$ be the coefficients of the terms $e^{\alpha x} x^p, \ldots, e^{\alpha x} x^{m+p}$ respectively.
- Then

$$z_i = \frac{b_{i-p} - \sum_{j=i+1}^{m+p} z_j C_{j-i+p} j^{\underline{j-i+p}}}{C_p i^{\underline{p}}}$$

$$= \frac{1}{C_p i^{\underline{p}}} \left(b_{i-p} - \frac{1}{(i-p)!} \sum_{j=i+1}^{m+p} (z_j \cdot j!) \cdot C_{j-i+p} \right)$$

- We can use CDQ divide and conquer FFT.

Finding Particular Solutions for a Class of Differential Equations

- Python Code Without FFT
- We can also use a similar method to find the particular solution of $\sum_{i=0}^{n} a_i y^{(i)} = P(x)\sin(\alpha x) + Q(x)\cos(\alpha x)$, but we need to note that differentiating $\sin(\alpha x)$ affects $\cos$, and vice versa.

References I

[1] Shun HUNG H.Architecture of 8-point Decimation-in-time FFT. [Online; created 16:58, 27 June 2011]. 2011. https://commons.wikimedia.org/wiki/File:DIT_N8_ALL.JPG.

[2] Wikipedia.Master theorem (analysis of algorithms) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Master%20theorem%20(analysis%20of%20algorithms)&oldid=1122572444. [Online; accessed 30-November-2022]. 2022.

[3] 毛啸. 再探快速傅里叶变换. 2016 年信息学奥林匹克竞赛中国国家队候选队员论文, 2016.

[4] 陈丹琦. 从《Cash》谈一类分治算法的应用. 2008 年信息学国家集训队作业, 2008.

References II

[5] Iostream.iostream 的博客 - 题解 P5850【calc 加强版】. 2020. https://www.luogu.com.cn/blog/user13052/solution-p5850.

[6] Wikipedia.Cayley–Hamilton theorem — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Cayley%E2%80%93Hamilton%20theorem&oldid=1120554488. [Online; accessed 01-December-2022]. 2022.

谢 谢