生成函数/多项式 算法、题目选讲

李白天（Entropy Increaser）

北京大学附属中学

转置原理

- 严格来说，转置原理不能无中生有得到一个算法，而是建立了一个问题的转化机制，使得我们得到了一个解决一个问题的算法之后，可以立即得到其转置问题的算法。
- 转置原理的断言是这样的：现有一个 $n \times m$ 的矩阵 $\mathbf{M}$，我们称一个算法可以对于任何一个输入的 m 维向量 $\mathbf{a}$ 计算出 $\mathbf{b} = \mathbf{M}\mathbf{a}$，那么可以将这一算法改写为其「转置算法」，用于在同样的时间复杂度内对于任何一个输入的 n 维向量 $\mathbf{b}$ 计算出 $\mathbf{a} = \mathbf{M}^{\mathsf{T}}\mathbf{b}$。
- 注意，此处虽然用的都是 $\mathbf{a}, \mathbf{b}$ 字母，但是两个算法不是「反解」的关系，而是「贡献反向」的关系。
- 转置原理是基于这样一个假设，算法运行过程中对输入的修改都是线性的，也就是应该不会出现需要计算 $a_i a_j$ 之类的情况。那么运算的实质就是将矩阵 $\mathbf{M}$ 分解为一系列基本运算的乘积 $\mathbf{M} = \mathbf{E}_r \cdots \mathbf{E}_2 \mathbf{E}_1$。
- 那么 $\mathbf{M}^{\mathsf{T}} = \mathbf{E}_1^{\mathsf{T}} \cdots \mathbf{E}_{r-1}^{\mathsf{T}} \mathbf{E}_r^{\mathsf{T}}$，转置算法就是将原算法倒序执行，并且每一步基本运算都进行转置。

- 让我们举一个最简单的例子。
- 前缀和问题：$b_i = \sum_{j \le i} a_j$，其转置为后缀和问题 $a_j = \sum_{j \le i} b_i$。

```
for (int i = 2; i <= n; ++i) a[i] += a[i - 1];
```

- 首先我们要将执行顺序倒序，变成

```
for (int i = n; i >= 2; --i) /* ... */
```

- 然后将 $a_i \leftarrow a_i + a_{i-1}$ 转置，就是 $a_{i-1} \leftarrow a_{i-1} + a_i$，因此转置算法就是

```
for (int i = n; i >= 2; --i) a[i - 1] += a[i];
```

基本操作

表格 1. 一些基本操作以及其转置

原操作	转置操作
$a_i \leftarrow a_i + c\,a_j$	$a_j \leftarrow a_j + c\,a_i$
$\mathrm{swap}(a_i, a_j)$	$\mathrm{swap}(a_i, a_j)$
$a_i \leftarrow a_j$	$a_j \leftarrow a_i$

DIF-DIT 的原理

- 通过转置原理，我们可以很方便地确认 DIF-DIT 的 FFT 实现是如何去掉 bitrev 操作的。
- 我们知道，设 n 阶单位根为 ω，那么 DFT 矩阵形如

$$\mathbf{DFT} = \begin{pmatrix} 1 & \omega & \omega^2 & \cdots \\ \omega & \ddots & \ddots & \ddots \\ \omega^2 & \ddots & \omega^{ij} & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}$$

- 这是个对称矩阵，也就有 $\mathbf{DFT}^{\mathrm{T}} = \mathbf{DFT}$，说明我们将计算 DFT 的算法转置，效果不变。

主流实现


```
for (int i = 0; i < lim; i++)
  if (i < rev[i])
    swap(f[i], f[rev[i]]);
for (int l = 1; l < lim; l <<= 1)
  for (int i = 0; i < lim; i += (l << 1))
    for (int j = 0; j < l; j++) {
      x = f[i + j];
      y = f[i + j + l] * w[j + l];
      a[i + j] = x + y;
      a[i + j + l] = x - y;
    }
```

- 让我们将这份代码转置。
- 转置使得 swap 的部分变到最后。
- 变量 l 应该从大到小。
- 代码中的 i, j 按照什么顺序都不影响。
- 原算法的核心步骤是 $\begin{pmatrix} 1 & \omega^j \\ 1 & -\omega^j \end{pmatrix} \begin{pmatrix} a_{i+j} \\ a_{i+j+l} \end{pmatrix}$，
- 转置为 $\begin{pmatrix} 1 & 1 \\ \omega^j & -\omega^j \end{pmatrix} \begin{pmatrix} a_{i+j} \\ a_{i+j+l} \end{pmatrix}$。

转置后的 FFT


```
for (int l = lim >> 1; l; l >>= 1)
  for (int i = 0; i < lim; i += (l << 1))
    for (int j = 0; j < l; j++) {
      x = f[i + j];
      y = f[i + j + l];
      a[i + j] = x + y;
     a[i + j + l] = (x - y) * w[j + l];
    }
for (int i = 0; i < lim; i++)
  if (i < rev[i])
    swap(f[i], f[rev[i]]);
```

- 考虑 IDFT 的时候还用原算法。
- 由于 DFT 的末尾和 IDFT 的开头各进行了一次位逆序置换，所以相当于没做。
- 这样一来，我们可以同时将两个过程中的位逆序操作删去。

新多点求值算法

- 相比于基于多项式取模的多点求值算法，新算法的优势在于其只需要进行一次多项式求逆，没有多项式取模。这使得多点求值实现的复杂程度得到减小，且常数大大减小。
- 首先让我们写出多点求值的原问题，这里为叙述方便，我们仅叙述 $n-1$ 次多项式取 n 处点值的情况。
- 设多项式 $f(x)=\sum_{i=0}^{n-1} f_i x^i$，需要对每个 $1\le i\le n$ 求出点值 $f(a_i)$。将 f_i 看做输入，得到矩阵

$$\begin{pmatrix} f(a_1) \\ \vdots \\ f(a_n) \end{pmatrix} = \begin{pmatrix} 1 & a_1 & \cdots & a_1^{n-1} \\ \vdots & \ddots & \ddots & \vdots \\ 1 & \cdots & \cdots & a_n^{n-1} \end{pmatrix} \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

- 观察其转置问题：$\begin{pmatrix} 1 & \cdots & 1 \\ a_1 & \ddots & a_n \\ \vdots & \ddots & \vdots \\ a_1^{n-1} & \cdots & a_n^{n-1} \end{pmatrix}\begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix}$
- 那么第 i 项就是 $\sum_{j=1}^{n} g_j a_j^i = \sum_{j=1}^{n} g_j [x^i] \frac{1}{1-a_j x} = [x^i] \sum_{j=1}^{n} \frac{g_j}{1-a_j x}$，因此我们的问题核心就是计算 $\sum_{j=1}^{n} \frac{g_j}{1-a_j x}$。
- 这个问题的做法是比较明朗的，我们过程中只需要先维护分式 $A(x)/B(x)$ 的形式，那么合并就是 $\frac{A_L(x)}{B_L(x)} + \frac{A_R(x)}{B_R(x)} = \frac{A_L(x)B_R(x) + A_R(x)B_L(x)}{B_L(x)B_R(x)}$，在最后求逆、乘法即可得到各项系数。过程的复杂度是 $\Theta(n \log^2 n)$。
- 这里要分清一个概念：只有 g 是这个**线性算法**的输入，只有分母 $A_L(x)B_R(x) + A_R(x)B_L(x)$ 的计算过程是在执行这个算法，而 $B_L(x)B_R(x)$ 其实是**预处理**过程。所以无论是转置算法还是原算法，都可以理解为事先完成预处理。

- 现在我们来描述转置算法。
- 首先完成预处理操作。依据分治结构建立一颗树，这棵树每个节点维护出区间内的 $B_{[l,r]}=\prod_{i=l}^{r}(1-a_i x)$。
- 剩下的就是线性算法的步骤了，第一步是将传入的多项式 $A(x)$ 转置乘以 $B_{[1,n]}(x)^{-1}$，当然转置乘的形式也就是俗称的「差卷积」。
- 然后开始将信息下传，就有 $A_R=\mathrm{mul}^{\mathsf{T}}(B_L)A, A_L=\mathrm{mul}^{\mathsf{T}}(B_R)A$。注意这里保留的次数应当和原问题对应，也就是 $\deg A_L=(\deg B_L)-1$，此时 $\mathrm{mul}^{\mathsf{T}}$ 需要的 DFT 长度只需要 $>A$ 就可以（从转置算法的角度也能应证）。
- 此时传到叶节点处应该都是常数，对应于所求的点值。

1s 1e6（顺便说两句）

- UOJ 500 这道题是一个有特殊性质的 2.5×10^5 次多项式 10^6 点求值问题，它被一般多点求值算法在 1s 内通过了。（negiizhao, skip2004, 2020.7）
- 除了诸如对 DFT 以及小范围暴力这种常规的底层优化手段，他们还进行了一些算法层面的优化。
- 若 $a_i^n\neq1$，那么发现 $(1+a_ix+\cdots+a_i^{n-1}x^{n-1})(1-a_ix)=1-a_i^nx^n\equiv1-a_i^n\pmod{x^n-1}$ 非常数，此时移项可得 $1+a_ix+\cdots+a_i^{n-1}x^{n-1}=\frac{1-a_i^n}{1-a_ix}\bmod(x^n-1)$。
- 这样一来形式幂乘法逆元也不需要求了，直接 DFT 之后各点求逆，也就是循环卷积逆就可以了。不过要特判掉 a_i 是单位根的情况，然后用一个 DFT 统一处理。

试看看！

练习 1. Do Use FFT（Gym 102978D）

给长为 N 的序列 A, B, C，对 $k = 1, \ldots, N$ 求出

$$\sum_{i=1}^{N} \left(C_i \times \prod_{j=1}^{k} (A_i + B_j) \right)$$

答案对 998244353 取模。保证 $N \le 2.5 \times 10^5$。

- 想要把问题转置，先要搞清楚把什么当成输入。
- 由于是线性变换，式子里 A, B 各自都有交叉项，那只能把 C 当成输入。
- 变换的矩阵即为 $M_{i,k} = \prod_{j=1}^{k} (A_i + B_j)$，转置问题即为

$$C_i = \sum_{k=1}^{N} \left(D_k \times \prod_{j=1}^{k} (A_i + B_j) \right)$$

- 记 $f(x) = \sum_{k=1}^{N} \left(D_k \times \prod_{j=1}^{k} (x + B_j) \right)$，那么这个问题就是对 $f(x)$ 多点求值。
- 那么我们只需要先求出 $f(x)$ 的表达式，对区间 $[l, r]$ 分治的时候同时维护 $\sum_{k=l}^{r} \left(D_k \times \prod_{j=l}^{k} (x + B_j) \right)$ 和 $\prod_{j=l}^{r} (x + B_j)$ 就好了。复杂度 $\Theta(n \log^2 n)$。

试看看！

练习 2. TREE & PRIME（IOI2021 集训队自选题 BY 陈峻宇）

给一颗 n 个节点的有根树，你到达节点 i 时会有 p_i 的概率获得一个金币。对于每个节点，请计算从 1 号点出生沿最短路走到它，有多大的概率获得的金币数量恰为素数。答案对 998244353 取模。

- 素数这个条件看起来就很具有一般性，我们考虑直接改写成：获得 k 个金币的收益是 a_k，对每个点求走到时候收益的期望。
- 那么转置问题就是给定一个序列 b，求出

$$\sum_i b_i \prod_{j \in \mathrm{Path}(1,i)} (p_j x + 1 - p_j)$$

- 如果是一条链，我们直接分治 FFT 就好做到了 $\Theta(n \log^2 n)$。
- 注意到一颗子树得到的多项式次数是子树的高度而不是大小，所以可以直接考虑长链剖分。一条长为 l 的链会在上层分治中花费 $O(l \log^2 n)$ 的代价，所以复杂度还是 $\Theta(n \log^2 n)$。

试看看！

练习 3. 「KROI2021」FEUX FOLLETS（LUOGU P7440，有改动）

给定序列 F，对于一个置换 π，其权值为 $F_{\mathrm{cyc}(\pi)}$，其中 $\mathrm{cyc}(\pi)$ 是置换中环的数量。

对于 $m=1,\ldots,n$，求出全体 m 阶错排的权值和，答案对 998244353 取模。

保证 $n\le 10^5$。

- 错排无非去掉大小为 1 的环。由于一个环的生成函数是 $\ln\frac{1}{1-x}$，去掉大小为 1 的环就是 $\ln\frac{1}{1-x}-x$。也就是说总共有 k 个环的 n 阶错排数应当为 $n![x^n]\frac{\left(\ln\frac{1}{1-x}-x\right)^k}{k!}$。
- 我们可以把问题写成计算 $F\left(\ln\frac{1}{1-x}-x\right), F(x)=\sum_k F_k\frac{x^k}{k!}$，但是这对计算没啥启发。
- 我们现在的问题是对每个 $m\le n$，计算 $\sum_k F_k A_{m,k}$，其中 $A_{m,k}=m![x^m]\frac{\left(\ln\frac{1}{1-x}-x\right)^k}{k!}$。
- 将它转置，我们考虑对于一组输入 G，对每个 $k\le n/2$ 计算 $\sum_m G_m A_{m,k}$。

- 我们先考虑怎么计算 $A_n(x)=\sum_k A_{n,k}x^k$。考虑二元生成函数 $\sum_n A_n(x)\frac{t^n}{n!}$。将其展开：

$$\sum_n A_n(x)\frac{t^n}{n!}=\sum_{n,k} A_{n,k}x^k\frac{t^n}{n!}$$

$$=\sum_{n,k}\left\{n![x^n]\frac{\left(\ln\frac{1}{1-x}-x\right)^k}{k!}\right\}x^k\frac{t^n}{n!}$$

$$=\sum_k x^k\frac{\left(\ln\frac{1}{1-t}-t\right)^k}{k!}$$

$$A(x,t)=\exp\left[x\left(\ln\frac{1}{1-t}-t\right)\right]$$

- 对 t 求导，有

$$\frac{\partial}{\partial t}A(x,t)=\left\{\exp\left[x\left(\ln\frac{1}{1-t}-t\right)\right]\right\}\cdot\frac{\partial}{\partial t}\left[x\left(\ln\frac{1}{1-t}-t\right)\right]$$

$$=A(x,t)\cdot x\frac{t}{1-t}$$

$$(1-t)\left[\frac{\partial}{\partial t}A(x,t)\right]=A(x,t)\cdot x\,t$$

$$[t^n](1-t)\left[\frac{\partial}{\partial t}A(x,t)\right]=[t^n]A(x,t)\cdot x\,t$$

$$\frac{A_{n+1}(x)}{n!}-\frac{A_n(x)}{(n-1)!}=\frac{A_{n-1}(x)}{(n-1)!}\cdot x$$

$$A_{n+1}(x)=n\,A_n(x)+n\,x\,A_{n-1}(x)$$

- 于是我们就得到了 $A_n=(n-1)A_{n-1}+(n-1)xA_{n-2}$。（当然也有组合解释，留作习题）
- 将其写作矩阵的形式，就有 $\begin{pmatrix}A_n\\A_{n-1}\end{pmatrix}=\begin{pmatrix}n-1 & (n-1)x\\1 & \end{pmatrix}\begin{pmatrix}A_{n-1}\\A_{n-2}\end{pmatrix}$，记这个转移矩阵为 M_n。
- 也就有 $A_m=(1\ 0)\mathbf{M}_m\mathbf{M}_{m-1}\cdots\mathbf{M}_1\begin{pmatrix}1\\0\end{pmatrix}$。因此求 $\sum_m G_mA_m(x)$ 只需求 $\sum_m G_m\cdot(\mathbf{M}_m\cdots\mathbf{M}_1)$。
- 那么我们还是分治就好了，对于一段 $[l,r]$ 需要维护出此时的 $\mathbf{M}_r\cdots\mathbf{M}_l$ 以及 $\sum_{m=l}^{r}G_m\cdot(\mathbf{M}_m\cdots\mathbf{M}_l)$。复杂度是 $\Theta(n\log^2 n)$。
- 综上，我们把上面这个巨大算法转置，就得到了原问题的一个 $\Theta(n\log^2 n)$ 解法。

新线性递推算法

- 给定数列 $a_0,\ldots,a_{k-1}$ 和 $b_1,\ldots,b_k$，在 a 上定义递推关系

$$a_n=\sum_{j=1}^{k} b_j a_{n-j}$$

- 一个众所周知的方法是通过计算 $x^n \bmod Q(x)$（$Q(x)=x^k-\sum_{j=1}^{k} b_j x^{k-j}$）在 $\Theta(k\log k\log n)$ 时间内计算 a_n。
- 但这种基于多项式取模的做法（Fiduccia, 1985）拥有较大的常数，接下来介绍一种常数较小，且不需要实现多项式求逆的算法。

- 和多项式取模不同的是，新算法通过考虑对有理分式 $[x^n]\frac{P(x)}{Q(x)}$ 的系数提取来优化算法。
- 由递推关系可知，$Q(x)=1-\sum_{j=1}^{k} b_j x^j$，以及 $P(x)=\left(\sum_{j=0}^{k-1} a_j x^j\right)\cdot Q(x) \bmod x^k$。
- 考虑如下等式：

$$\frac{P(x)}{Q(x)}=\frac{P(x)Q(-x)}{Q(x)Q(-x)}$$

- 注意到 $Q(x)Q(-x)$ 是偶函数，不妨设其为 $V(x^2)=Q(x)Q(-x)$，再设 $U_0(x^2)+xU_1(x^2)=P(x)Q(-x)$。（也就是将 $P(x)Q(-x)$ 的次数拆成奇数和偶数部分）

- 那么我们提取系数 $[x^n]\frac{U_0(x^2)}{V(x^2)}+x\frac{U_1(x^2)}{V(x^2)}$ 的时候，根据二进制最低位递归至一侧即可，也即

$$[x^n]\frac{P(x)}{Q(x)}=[x^{\lfloor n/2\rfloor}]\frac{U_{n\bmod 2}(x)}{V(x)}$$

- 依此迭代，便能在 $\log_2 n$ 轮内将 n 降至 0，此时 $P(0)/Q(0)$ 就是答案。
- 这一算法其实对于 FFT 来说非常友好，可以通过一些更细致的优化让整个做法的时间变成上述粗糙实现（每轮进行两次长为 k 的多项式乘法）的 $1/3$，不过这里不是卡常课就不展开啦。

反哺

- 在 n 确定的情况下，已知 $a_0,\ldots,a_{k-1}$，求 a_n，其实就是 $(a_0,\ldots,a_{k-1})\mapsto a_n$ 的一个线性变换。
- 这个变换其实就是一个向量，它正是 $x^n \bmod Q(x)$ 的系数。
- 因此我们可以直接将新线性递推算法的过程转置，就得到了一个不需要多项式求逆的计算 $x^n \bmod Q(x)$ 的方法。

试看看！

练习 4. MAKING CHANGE（CODECHEF CHANGE），有改动

有 n 种硬币，第 i 种面值是 D_i 元，每种都有无限个。求拼出 C 的方案数取模 M。

保证 $n \le 50, D_i \le 500, C \le 10^{100}, M \le 2^{63}$。

- 初始状态是求 $[x^C]\frac{1}{(1-x^{D_1})(1-x^{D_2})\cdots(1-x^{D_n})}$。
- 设为 $[x^C]\frac{P(x)}{\prod_{i=1}^{n}(1-x^{D_i})}$，考虑上下同时乘以 $\prod_{i=1}^{n}(1+x^{D_i})$，就变成了

$$\frac{P(x)}{\prod_{i=1}^{n}(1-x^{D_i})}=\frac{P(x)\prod_{i=1}^{n}(1+x^{D_i})}{\prod_{i=1}^{n}(1-x^{2D_i})}$$

- 因此每次只需要给 $P(x)$ 每次乘以一个 $(1+x^{D_i})$，P 的次数不超过 nD，因此迭代的每一轮都是 $\Theta(n^2D)$ 的。
- 不算进制转换的时间的话，复杂度为 $\Theta(n^2D\log C)$。

试看看！

练习 5. 王的象棋世界（LIBREOJ 6738）

有一个 $R \times C$ 的棋盘，你有 Q 组询问，每次询问国王走 $R-1$ 步从 $(1,a)$ 到达 (R,b) 有多少种方案。答案对 998244353 取模。

保证 $C \le 10^5, C \le R \le 10^9, Q \le 10^5$。

- 首先列出递推式，易见转移可以写作一个矩阵

$$\mathbf{M}=\begin{pmatrix} 1 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 1 \end{pmatrix}$$

- 每次询问的就是 $(\mathbf{M}^{R-1})_{a,b}$。
- 我们考虑 $\mathbf{M}$ 的特征多项式 $p(\lambda)$，**Cayley-Hamilton 定理**告诉我们 $p(\mathbf{M})=\mathbf{O}$。
- 设 $\lambda^{R-1}\bmod p(\lambda)=\sum_{r=0}^{c-1}c_r\lambda^r$，也就有 $\mathbf{M}^{R-1}=\sum_{r=0}^{c-1}c_r\mathbf{M}^r$，两边取第 a 行第 b 列，就得到了如下结果：
- $f(R,a,b)=\sum_{r=1}^{c}c_{r-1}f(r,a,b)$，其中 $f(r,a,b)$ 是从 $(1,a)$ 走到 (r,b) 的方案数。

- 有边界的路径计数可以使用经典的反射容斥解决，由于现在 $r \le C$，可知一条路径顶多只在一侧超出边界。
- 因此这里设 $g(r,d) = [x^d](x^{-1}+x^0+x^1)^{r-1}$ 为走 $r-1$ 步，最后横向移动的总和为 d 的方案数。就有 $f(r,a,b) = g(r,a-b) - g(r,a-(-b)) - g(r,a-(2(C+1)-b))$。
- 将其写作 $([x^{a-b}] - [x^{a-(-b)}] - [x^{a-(2(C+1)-b)}])(x^{-1}+x^0+x^1)^{r-1}$，由于还要对 c_r 求和，我们需要计算出

$$\sum_{r=0}^{C-1} c_r (x^{-1}+x^0+x^1)^r$$

- 只需分治便可做到 $\Theta(C \log^2 C)$。每个询问可以 $\Theta(1)$ 回答。

- 最后解决一下特征多项式的计算，设 C 阶情形的特征多项式为 p_C，观察形式

$$p_C(\lambda) = \det(\lambda \mathbf{I} - \mathbf{M}) = \det \begin{pmatrix} \lambda - 1 & -1 & & \\ -1 & \lambda - 1 & \ddots & \\ & \ddots & \ddots & \end{pmatrix}$$

- 将第一项展开发现 $p_C = (\lambda - 1)p_{C-1} - p_{C-2}$。
- 其实系数是可以 $\Theta(C)$ 算的，但是由于不是瓶颈，这里就给出一个简单的 $\Theta(C \log C)$ 做法。
- 将 λ 带入一个值，可以 $\Theta(\log C)$ 时间求出答案，那么考虑将所有单位根带入，然后再做一次 IDFT 就可以了。
- 整个问题的瓶颈在于 $\lambda^{R-1} \bmod p(\lambda)$ 的计算，复杂度为 $\Theta(C \log C \log R + Q)$。

谢谢大家

预祝大家省选顺利