中国计算机学会
China Computer Federation

IOI 题目选讲与矩阵行列式

绍兴市第一中学　王展鹏

2021 年 2 月

连接擎天树 supertrees(D1T2)

中国计算机学会 China Computer Federation

要求构造一个点数为 n 的无向图 (或者判断无解)，满足:
任意给定 $0 \leq i, j \leq n-1$，恰好有 $p_{i,j}$ 条从塔 i 到塔 j 的不同路径，其中 $0 \leq p_{i,j} \leq 3$。
$1 \leq n \leq 100, p_{i,i} = 1, p_{i,j} = p_{j,i}, 0 \leq p_{i,j} \leq 3.$
子任务 1(11 分)：$p_{i,j} = 1$。
子任务 2(10 分)：$p_{i,j} = 0/1$。
子任务 3(19 分)：$p_{i,j} = 0/2 (i \neq j)$。
子任务 4(35 分)：$p_{i,j} \leq 2$, 并且至少有一种建设方案满足要求。
子任务 5(4 分)：$p_{i,j} \leq 2$.
子任务 6(21 分)：没有附加限制条件。

子任务一：$p_{i,j} = 1$

- 随意构造一棵树即可。

子任务二：$p_{i,j} = 0/1$

- 如果 $p[i][j] = 0$，说明两个点不联通，我们可以对于每一个联通块单独考虑，对于每一个联通块，显然有 $p[i][j] > 0$。具体实现可以用并查集维护联通块，最后判断一下所有 $p[i][j]$ 是否合法。

子任务二：$p_{i,j} = 0/1$

- 如果 $p[i][j] = 0$，说明两个点不联通，我们可以对于每一个联通块单独考虑，对于每一个联通块，显然有 $p[i][j] > 0$。具体实现可以用并查集维护联通块，最后判断一下所有 $p[i][j]$ 是否合法。
- 对于每一个联通块，由于 $p_{i,j} = 1$，随意构造一棵树就可以了。

子任务三：$p_{i,j} = 0/2 (i \neq j)$

- 仍然对于每一个联通块判断是否有解，由于 $p_{i,j} = 2$, 所以直接构造一个环就可以了，注意当一个联通块大小为二时无解。

子任务四，五：$p_{i,j} \leq 2$

- 仍然对于每一个联通块判断是否有解。如果存在 $p_{i,j} = 1 (i \neq j)$, 则 i, j 只有一条路径，容易证明，在合法的图中，对于任意 k，我们都有 $p[i][k] = p[j][k]$。也就是说，我们可以将那些两两之间只有一条路径的点“合并”，之后都可以将它们看成一个点 (也就是“缩点”过程)。

子任务四，五：$p_{i,j} \leq 2$

- 仍然对于每一个联通块判断是否有解。如果存在 $p_{i,j} = 1 (i \neq j)$, 则 i, j 只有一条路径，容易证明，在合法的图中，对于任意 k，我们都有 $p[i][k] = p[j][k]$。也就是说，我们可以将那些两两之间只有一条路径的点“合并”，之后都可以将它们看成一个点 (也就是“缩点”过程)。
- 剩余的点之间就只有 $p_{i,j} = 2$ 了，只要将每个联通块取出一个“代表”来，将这些代表连成一个环就可以了。

子任务六：$p_{i,j} \leq 3$

- 最后一个子任务只有四分，考虑到这是 IOI 不是 NOI，因此只要出现了 $p_{i,j} = 3$ 直接判断无解再结合子任务五的算法就可以通过这个子任务了。

子任务六：$p_{i,j} \leq 3$

- 最后一个子任务只有四分，考虑到这是 IOI 不是 NOI，因此只要出现了 $p_{i,j} = 3$ 直接判断无解再结合子任务五的算法就可以通过这个子任务了。
- 稍加分析画图可知，一旦出现 $p_{i,j} = 3$ 的点对，那必然也存在 $p_{i,j} = 4$ 的点对，而这与数据限制不符。

嘉年华奖券 tickets(D1T3)

假设共有 n 种颜色的奖券，每张奖券涂上了其中的一种颜色并且印上了一个非负整数。不同奖券上的数字可能相同。n 保证是偶数。

每种颜色的奖券有 m 张，也就是说他共有 $n \cdot m$ 张奖券。其中，第 i 种颜色对应的第 j 张奖券上印的数字为 $x_{i,j}$。

一次奖券游戏要进行 k 轮，轮次的序号从 0 到 $k-1$。每一轮按照下面的方式进行：

- 首先，从每种颜色的奖券中各选出一张，形成一个 n 张奖券的集合。
- 随后，游戏负责人记录下这个集合中奖券上的数字 $a_0, a_1, \ldots, a_{n-1}$。不需要考虑这 n 个整数的顺序。
- 接下来，游戏负责人从一个幸运抽奖箱中抽取一张特殊卡片，上面印有整数 b。
- 对于上述集合中每一个奖券上的数字 $a_i (0 \leq i \leq n-1)$，游戏负责人会计算 a_i 和 b 的差的绝对值。让 S 代表这 n 个差的绝对值之和。
- 所得到的数字 S 就是本轮能够获得的奖励数额。
- 一轮游戏结束后，本轮集合中的奖券全部被丢弃，不会在未来的轮次所使用。

实际上，幸运抽奖箱里面内置了一台打印机。在每一轮，游戏负责人首先找到一个能够最小化当前轮次游戏奖励的整数 b，然后将该数字打印在他所抽取的特殊卡片上。

知道了这些信息之后，要求设计每轮游戏中的奖券分配方案，使得 k 轮游戏中获得的总体奖励数额之和最大。

$2 \leq n \leq 1500$ 且 n 为偶数,$1 \leq k \leq m \leq 1500$,$0 \leq x_{i,j} \leq 10^9$.

子任务 1(11 分)：$m = 1$.

子任务 2(16 分)：$k = 1$.

子任务 3(14 分)：$0 \leq x_{i,j} \leq 1$。

子任务 4(14 分)：$k = m$.

子任务 5(12 分)：$n, m \leq 80$.

子任务 6(23 分)：$n, m \leq 300$.

子任务 7(10 分)：没有其他限制。

子任务一：$m = 1$

- 分配的方案已经确定，我们只需要确定 b 的取值就可以了，根据初中学过的知识，b 取每一轮数字的中位数即可最小化奖励数额。

子任务二：$k = 1$

- 观察奖励数额的式子 $\sum |a_i - b|$，由于 b 是中位数且 n 是偶数，那么必然有一半的 a_i 较大的项取正，其余取负。故直接选择最大的 $n/2$ 个数以及最小的 $n/2$ 个数即可。

子任务四-六

- 我们断言，最优答案是：n 种颜色卡片中各选 k 张，其中一半 $nk/2$ 选择正权，另一半选择负权，且任意正权权值绝对值大于负权权值的绝对值。

子任务四-六

中国计算机学会 China Computer Federation

- 我们断言，最优答案是：n 种颜色卡片中各选 k 张，其中一半 $nk/2$ 选择正权，另一半选择负权，且任意正权权值绝对值大于负权权值的绝对值。
- 不难发现这些条件都是必要条件，故这是答案的上界，且这个答案也是可以达到的，每次从这选定的 nk 张卡牌在每种颜色中选出 $n/2$ 张正权，$n/2$ 张负权卡片，将问题规模缩小后递归就能达到这个上界。

子任务四-六

- 我们断言，最优答案是：n 种颜色卡片中各选 k 张，其中一半 $nk/2$ 选择正权，另一半选择负权，且任意正权权值绝对值大于负权权值的绝对值。
- 不难发现这些条件都是必要条件，故这是答案的上界，且这个答案也是可以达到的，每次从这选定的 nk 张卡牌在每种颜色中选出 $n/2$ 张正权，$n/2$ 张负权卡片，将问题规模缩小后递归就能达到这个上界。
- 想要找方案有两种方法，可以凸优化，也可以直接贪心。贪心可以假设开始选了 nk 张负权卡片，然后进行 $nk/2$ 次贪心将负权换为正权的操作。

装饼干 biscuits(D2T1)

有 k 种不同类型的饼干，编号为从 0 到 $k-1$。类型为 i（$0 \le i \le k-1$）的每块饼干都有一个口味值 2^i。在食品储藏室里，有 a_i 块类型为 i 的饼干。

对每种类型的饼干，要求在每个袋子都会装上 0 或者多块。所有袋子里面类型为 i 的饼干的总块数不能超过 a_i。一个袋子里面所有饼干的口味值的总和，被称为这袋饼干的总口味值。

求存在多少不同的 y 值，使得可以装出 x 袋饼干，而且每袋饼干的总口味值都等于 y。有 q 组数据。

$1 \le k \le 60, 1 \le q \le 1000, 1 \le x \le 10^{18}, 0 \le a_i \le 10^{18}$，对于每组数据，食物储藏室里所有饼干的口味值总和都不会超过 10^18。

子任务 1(9 分)：$q \le 10$，且对于每组数据，食物储藏室里所有饼干的口味值总和都不会超过 100000。

子任务 2(12 分)：$x = 1, q \le 10$。

子任务 3(21 分)：$x \le 10000, q \le 10$。

子任务 4(35 分)：对于每组数据，正确的返回结果都不会超过 200000。

子任务 5(23 分)：没有附加限制条件。

子任务一：$q \leq 10$，且对于每组数据，食物储藏室里所有饼干的口味值总和都不会超过 100000

- 朴素地判断一下每一个 y 是否合法，显然可以按照饼干口味值从大到小，能装就装的策略是最优的。

子任务四：对于每组数据，正确的返回结果都不会超过 200000

- 考虑每一个 y 是否合法。从低位往高位判断，需要对于每一个 i，要求只靠口味值 $\leq 2^i$ 的饼干都可以装出 y 的 $0-i$ 位，即：

$$\sum 2^i a_i / x \geq y \ and \ (2^{i+1}-1)$$

子任务四：对于每组数据，正确的返回结果都不会超过 200000

- 考虑每一个 y 是否合法。从低位往高位判断，需要对于每一个 i，要求只靠口味值 $\leq 2^i$ 的饼干都可以装出 y 的 $0-i$ 位，即：

$$\sum 2^i a_i/x \geq y \ and \ (2^{i+1}-1)$$

- 换言之，要求对于 y 二进制分解从低位到高位任意一个前缀，都有一个上界限制 lim_i. 从高位往低位搜索 y 的每一位取值，设当前决定第 i 位的取值，则当前取值需要满足所有 lim_j 的限制 $(j \geq i)$。

子任务四：对于每组数据，正确的返回结果都不会超过 200000

- 考虑每一个 y 是否合法。从低位往高位判断，需要对于每一个 i，要求只靠口味值 $\leq 2^i$ 的饼干都可以装出 y 的 $0-i$ 位，即：

$$\sum 2^i a_i/x \geq y \ and \ (2^{i+1}-1)$$

- 换言之，要求对于 y 二进制分解从低位到高位任意一个前缀，都有一个上界限制 lim_i. 从高位往低位搜索 y 的每一位取值，设当前决定第 i 位的取值，则当前取值需要满足所有 lim_j 的限制 $(j \geq i)$。
- 由于数据范围限制的比较紧 (还有 1000 组数据)，预处理时需要删掉永远不会选 1 的位，搜索时保留最紧的那个限制并根据限制快速跳到一个能选 1 的位置才能保证复杂度 (决策树是一颗度要么为 0 要么为 2 的二叉树) 。

其余子任务

- 最后一步就是经典的数位 DP 了，把搜索换成 DP，记录当前位以及最紧限制，枚举 y 当前位选 $0/1$ 转移，复杂度 $O(qk^2)$。

众所周知的定义

这里假设大家至少掌握矩阵快速幂的相关知识。

定义 (行列式)

方阵 A 的行列式定义为 $|A| = \sum_{\sigma}(-1)^{sgn(\sigma)} \prod A_{i,\sigma_i}$。$\sigma$ 是一个排列，sgn 表示逆序对数。

定义 (分块矩阵与乘法)

对于一个矩阵，以水平线和垂直线将其划分为若干更小的矩阵称为分块矩阵。分块矩阵中，位在同一行（列）的每一个子矩阵，都拥有相同的列数（行数）。左矩阵的列分块完全和右矩阵的行分块相同时，可以做分块乘法。规则类似于普通的矩阵乘法，相当于将每一块矩阵看作一个元素，元素乘法对应矩阵乘法。

高斯消元

- 求解一般域上行列式，最常见的的算法是高斯消元法，原理是行列式有如下性质：
  1. 在行列式中，一行（列）元素全为 0，则此行列式的值为 0；
  2. 行列式中的两行（列）互换，改变行列式正负符号；
  3. 在行列式中，某一行（列）有公因子 k，则可以提出 k；
  4. 将一行（列）的 k 倍加进另一行（列）里，行列式的值不变。

  这些性质由定义不难证明，由这些性质，可以将原矩阵行列式转化成上三角矩阵求解，时间复杂度 $O(n^3)$。
  行列式还有另外一条经典性质：

定理 (行列式的乘法定理)

对于任意两个 $n \times n$ 矩阵 A, B，有

$$\det(AB) = \det(A)\det(B)$$

用矩阵乘法计算行列式

- 我们知道，矩阵乘法可以在低于 $O(n^3)$ 的复杂度内计算，现在最优的复杂度是 $O(n^\omega)(\omega \approx 2.373)$，这个算法可以见参考文献 [4]。由于此算法过于复杂这里就不做介绍了，我们更关心如何只用矩阵乘法来计算行列式。传统方法是对于矩阵 $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$，有：

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & D - C \times A^{-1}B \end{bmatrix}$$

所以

$$\begin{aligned} \det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) &= \det\left(\begin{bmatrix} A & 0 \\ C & D - C \times A^{-1}B \end{bmatrix}\right) \\ &= \det(A)\det(D - C \times A^{-1}B) \end{aligned}$$

用矩阵乘法计算行列式

- 我们知道，矩阵乘法可以在低于 $O(n^3)$ 的复杂度内计算，现在最优的复杂度是 $O(n^\omega)(\omega \approx 2.373)$，这个算法可以见参考文献 [4]。由于此算法过于复杂这里就不做介绍了，我们更关心如何只用矩阵乘法来计算行列式。传统方法是对于矩阵 $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$，有：

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & D - C \times A^{-1}B \end{bmatrix}$$

所以

$$\begin{aligned} \det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) &= \det\left(\begin{bmatrix} A & 0 \\ C & D - C \times A^{-1}B \end{bmatrix}\right) \\ &= \det(A)\det(D - C \times A^{-1}B) \end{aligned}$$

- 然而即使原矩阵可逆，矩阵 A 也不一定可逆 (例如每行每列都恰好只有一个 1 其余都是 0 的矩阵)，为此往往需要采用一些随机等方法，这里我们介绍一个确定的，适用于一般域上的行列式算法。

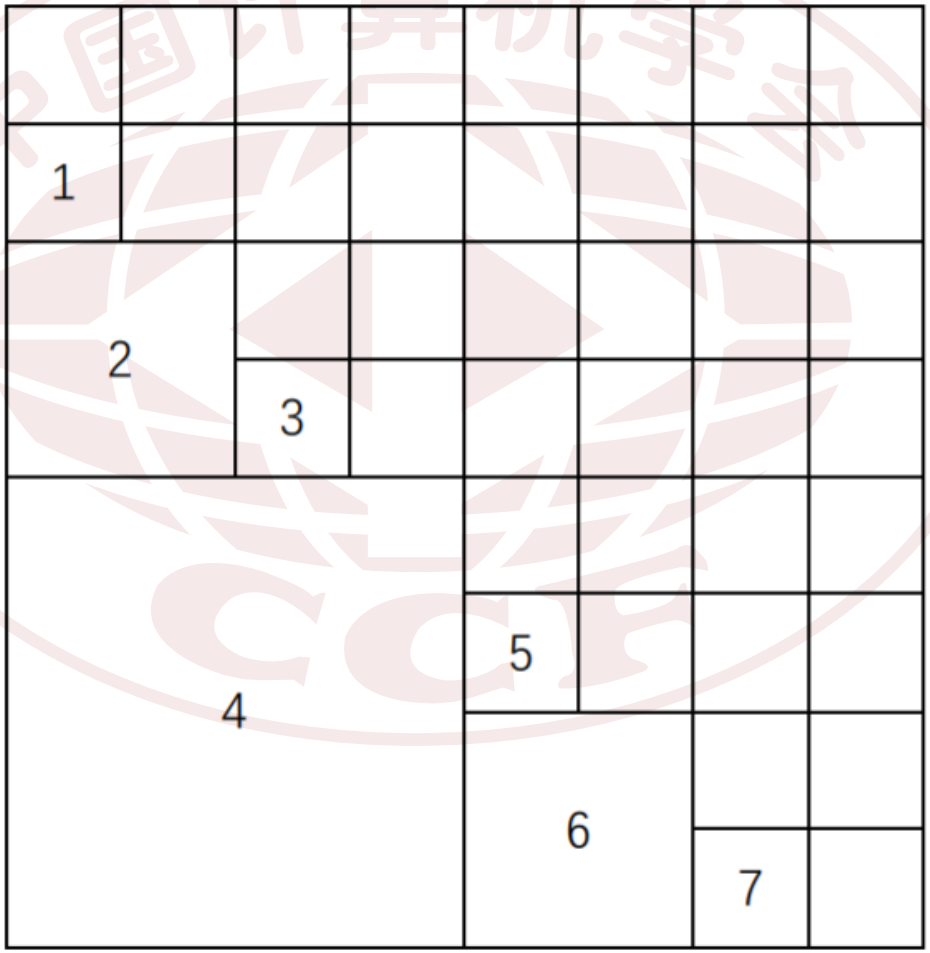
用矩阵乘法计算行列式

- 事实上，我们想要解决的是更一般的问题，即将可逆矩阵 M 分解为：$M = LUP$ 的形式，其中 L 是对角线上元素为 1 的下三角矩阵，U 是上三角矩阵，P 是排列矩阵 (即每行每列恰好一个 1，其余元素为 0)。

用矩阵乘法计算行列式

- 事实上，我们想要解决的是更一般的问题，即将可逆矩阵 M 分解为：$M = LUP$ 的形式，其中 L 是对角线上元素为 1 的下三角矩阵，U 是上三角矩阵，P 是排列矩阵 (即每行每列恰好一个 1，其余元素为 0)。
- 具体算法为维护矩阵 M_i，其中左上角 $i \times i$ 部分为上三角矩阵，对于矩阵 M_i，将一个小正方形消成 0，再将一个非零元素通过交换列的方式换到 $i+1, i+1$ 的位置，即可将矩阵 M_i 转化为 M_{i+1}。

- 直接用表格的形式更好理解一些。表格中的数字 i 表示是在 M_i 的基础上用上边的上三角矩阵消除的。具体消除一个矩阵的方法可以通过左乘一个下三角分块矩阵实现。由矩阵性质，可以保证 $i+1$ 行有非零元素，将其换到第 $i+1$ 列即可。



具体细节与时间复杂度

- 消元过程对应了左乘一个下三角分块矩阵，交换列过程对应了右乘一个排列矩阵，因此有 $M_{i+1} = L_i M_i P_i$，最终得到 M_n 即一个 $n \times n$ 的上三角矩阵，故有 $M_n = L_{n-1} L_{n-2} \cdots L_0 M P_0 P_1 \cdots P_{n-1}$，化简一下就得到了 $U = LMP$，求个逆得到 $M = L' U P'$.

具体细节与时间复杂度

- 消元过程对应了左乘一个下三角分块矩阵，交换列过程对应了右乘一个排列矩阵，因此有 $M_{i+1} = L_i M_i P_i$，最终得到 M_n 即一个 $n \times n$ 的上三角矩阵，故有 $M_n = L_{n-1} L_{n-2} \cdots L_0 M P_0 P_1 \cdots P_{n-1}$，化简一下就得到了 $U = LMP$，求个逆得到 $M = L' U P'$.
- 瓶颈显然在消元过程，对于矩阵 M_i，消元过程是一个 $2^s \times 2^s$ 矩阵和 $2^s \times n$ 矩阵的乘法 (过程中有求逆但不是瓶颈)，将右边的矩阵分成 $n/2^s$ 块 (其中 $s = lowbit(i)$)，复杂度是 $O((2^s)^\omega (n/2^s))$. 不难计算总的复杂度是 $O(n^\omega)$.

具体细节与时间复杂度

- 消元过程对应了左乘一个下三角分块矩阵，交换列过程对应了右乘一个排列矩阵，因此有 $M_{i+1} = L_i M_i P_i$，最终得到 M_n 即一个 $n \times n$ 的上三角矩阵，故有 $M_n = L_{n-1} L_{n-2} \cdots L_0 M P_0 P_1 \cdots P_{n-1}$，化简一下就得到了 $U = LMP$，求个逆得到 $M = L'UP'$.
- 瓶颈显然在消元过程，对于矩阵 M_i，消元过程是一个 $2^s \times 2^s$ 矩阵和 $2^s \times n$ 矩阵的乘法 (过程中有求逆但不是瓶颈)，将右边的矩阵分成 $n/2^s$ 块 (其中 $s = lowbit(i)$)，复杂度是 $O((2^s)^\omega (n/2^s))$. 不难计算总的复杂度是 $O(n^\omega)$.
- 综上，在任意域上的矩阵行列式可以做到矩阵乘法的复杂度。顺带一提，这个算法顺便也算出了矩阵的逆。

位复杂度

- 之前我们讨论了一些行列式的算法，但讨论复杂度时都是将元素之间的加减乘除视为 $O(1)$。但即使在最朴素的有理数域上的行列式，假设开始矩阵元素均为整数且大小视为 $O(1)$. 在行列式计算过程中元素规模马上会变得非常大。

位复杂度

- 之前我们讨论了一些行列式的算法，但讨论复杂度时都是将元素之间的加减乘除视为 $O(1)$。但即使在最朴素的有理数域上的行列式，假设开始矩阵元素均为整数且大小视为 $O(1)$. 在行列式计算过程中元素规模马上会变得非常大。
- 因此，我们考虑如果输入矩阵元素均为整数且规模视为 $O(1)$，要计算行列式的精确值能做到多低的复杂度。

- 由于我们已经有了一般域上的 $O(n\omega)$ 行列式算法，将其应用到模素数域上能做到同样的复杂度，要将不取模的情况转化为素数，让人联想到中国剩余定理。
  令最终行列式位数级别为 $g(n)$，根据定义，容易发现 $g(n)$ 不超过 $\tilde{O}(n)$，我们需要用 $O(g(n))$ 个大质数来还原答案，每次复杂度 $O(n^\omega)$，再结合插值，总的复杂度 $\tilde{O}(n^{\omega+1})$。

解线性方程组

- 我们自然不满足于此，不过在计算行列式之前，有一个更加基础的问题：解线性方程组 $Ax = b$(矩阵 A 可逆)。

解线性方程组

- 我们自然不满足于此，不过在计算行列式之前，有一个更加基础的问题：解线性方程组 $Ax = b$(矩阵 A 可逆)。
- 作为之后行列式算法的基础，我们先考虑这个问题如何快速求解(计算位复杂度)。

p-adic lifting

- 选取素数 p 使得矩阵 A 在模 p 的域下可逆 (不难证明等价于 p 与行列式互质，因此 p 规模为 $O(1)$)。

p-adic lifting

- 选取素数 p 使得矩阵 A 在模 p 的域下可逆 (不难证明等价于 p 与行列式互质，因此 p 规模为 $O(1)$)。
- 倍增计算 $C_i = A^{-1} \bmod p^{2^i}$，类似牛顿迭代，有等式：

$$C_i = C_{i-1} - (C_{i-1}((AC_{i-1} - I))) \bmod p^{2^i}$$

p-adic lifting

- 选取素数 p 使得矩阵 A 在模 p 的域下可逆 (不难证明等价于 p 与行列式互质，因此 p 规模为 $O(1)$)。
- 倍增计算 $C_i = A^{-1} \bmod p^{2^i}$，类似牛顿迭代，有等式：

$$C_i = C_{i-1} - (C_{i-1}((AC_{i-1} - I))) \bmod p^{2^i}$$

- 设最终 $C = A^{-1} \bmod p^m$，则本部分复杂度为 $O(n^\omega)$.

还原取模

- 令 $x_0 = Cb$ 则 $x_0 = x \bmod p^m$.

还原取模

- 令 $x_0 = Cb$ 则 $x_0 = x \bmod p^m$.
- 也就是说，我们现在得到一个有理数取模后的结果，想要还原这个有理数，这里不加证明地给出一个 $O(m^2)$ 的类似于扩展欧几里得的算法。

还原取模

- 令 $x_0 = Cb$ 则 $x_0 = x \bmod p^m$.
- 也就是说，我们现在得到一个有理数取模后的结果，想要还原这个有理数，这里不加证明地给出一个 $O(m^2)$ 的类似于扩展欧几里得的算法。
- 设 $sg + tp^m = f,\ f/g$ 是原有理数，设：

$$u_0 = p^m, u_1 = s, v_0 = 0, v_1 = 1$$

$$q_i = \lfloor u_{i-2}/u_{i-1} \rfloor, u_i = u_{i-2} - q_i u_{i-1}, v_i = v_{i-2} + q_i v_{i-1}.$$

还原取模

- 令 $x_0 = Cb$ 则 $x_0 = x \bmod p^m$.
- 也就是说，我们现在得到一个有理数取模后的结果，想要还原这个有理数，这里不加证明地给出一个 $O(m^2)$ 的类似于扩展欧几里得的算法。
- 设 $sg + tp^m = f,\ f/g$ 是原有理数，设：

$$u_0 = p^m,\ u_1 = s,\ v_0 = 0,\ v_1 = 1$$

$$q_i = \lfloor u_{i-2}/u_{i-1} \rfloor,\ u_i = u_{i-2} - q_i u_{i-1},\ v_i = v_{i-2} + q_i v_{i-1}.$$

- 取最小的 t 使得 $u_t < p^{m/2}$，则 $u_t/v_t = f/g$. 在本问题中，取 $m = O(n \log n)$ 即可。

还原取模

- 令 $x_0 = Cb$ 则 $x_0 = x \bmod p^m$.
- 也就是说，我们现在得到一个有理数取模后的结果，想要还原这个有理数，这里不加证明地给出一个 $O(m^2)$ 的类似于扩展欧几里得的算法。
- 设 $sg + tp^m = f,\ f/g$ 是原有理数，设：

$$u_0 = p^m,\ u_1 = s,\ v_0 = 0,\ v_1 = 1$$

$$q_i = \lfloor u_{i-2}/u_{i-1} \rfloor,\ u_i = u_{i-2} - q_i u_{i-1},\ v_i = v_{i-2} + q_i v_{i-1}.$$

- 取最小的 t 使得 $u_t < p^{m/2}$，则 $u_t/v_t = f/g$. 在本问题中，取 $m = O(n \log n)$ 即可。
- 由于要还原 n 次，算上求逆，总的复杂度是 $\tilde{O}(n^3)$.

史密斯标准型

定义 (史密斯标准型)

对于任意可逆整数矩阵 A，存在矩阵 X, Y 满足 $|\det(X)| = |\det(Y)| = 1$，使得 $A = XSY$ 成立，其中 $S = diags_n, \cdots, s_1$，且满足 $s_i \in \mathbb{Z}, s_i | s_{i+1}$，则称 S 是 A 的史密斯标准型。所有 s_i 称为不变因子。

史密斯标准型

定义 (史密斯标准型)

对于任意可逆整数矩阵 A，存在矩阵 X, Y 满足 $|\det(X)| = |\det(Y)| = 1$，使得 $A = XSY$ 成立，其中 $S = diags_n, \cdots, s_1$，且满足 $s_i \in \mathbb{Z}, s_i | s_{i+1}$，则称 S 是 A 的史密斯标准型。所有 s_i 称为不变因子。

- 因此，我们只需要算出所有不变因子，就可以算出行列式的绝对值了。然而，计算不变因子并不简单，为此需要许多引理，碍于时间有限，大部分引理我们都不再证明，想要证明的选手可以阅读参考文献 [1]。

最大不变因子

- 设计一个足够大的随机边界 $M = O(n \log n)$，多次随机 n 维列向量 b，解方程 $Ax = b$, 取 y_i 为 x_i 分母，取 y_i，将此过程执行多次并取所有 y_i 的 lcm，有引理：

最大不变因子

- 设计一个足够大的随机边界 $M = O(n \log n)$，多次随机 n 维列向量 b，解方程 $Ax = b$, 取 y_i 为 x_i 分母，取 y_i，将此过程执行多次并取所有 y_i 的 lcm，有引理：

引理

对于上述算法，执行两次随机过程后得到的 *lcm* 至少有 $1/3$ 的概率是 A 的最大不变因子。

最大不变因子

- 设计一个足够大的随机边界 $M = O(n \log n)$，多次随机 n 维列向量 b，解方程 $Ax = b$，取 y_i 为 x_i 分母，取 y_i，将此过程执行多次并取所有 y_i 的 lcm，有引理：

引理

对于上述算法，执行两次随机过程后得到的 *lcm* 至少有 $1/3$ 的概率是 A 的最大不变因子。

- 因此根据此引理，只要多随机几次就可以有一个很大概率满足 lcm 的最大不变因子了。

第 k 大不变因子

中国计算机学会 China Computer Federation

引理

对于秩不超过 k 的矩阵 B，设 $A+B$ 从小到大的不变因子为 $t_1, \cdots, t_n$，则满足 $s_i | t_{i+k}$.

第 k 大不变因子

引理

对于秩不超过 k 的矩阵 B，设 $A+B$ 从小到大的不变因子为 $t_1, \cdots, t_n$，则满足 $s_i | t_{i+k}$.

引理

取足够大的 $M_2 = C_2 n^2 \log n, M_1 = C_1 n^2 M_2$. 则随机元素属于 $[0, M_1 - 1]$ 的规模 $n \times k$ 矩阵 $U_i (1 \le i \le 15)$ 和规模 $k \times n$ 矩阵 $V_i (1 \le i \le 15)$. 将 M_1 换成 M_2，其余用同样的方法生成 $U_{16}, V_{16}.$，则 $\gcd\{s_n, t_{1,n}, \cdots, t_{16,n}\} = s_{n-k}$ 的概率至少有 $1/2$. 其中 $t_{i,n}$ 表示 $A + U_i V_i$ 的最大不变因子。

第 k 大不变因子

中国计算机学会 China Computer Federation

引理

对于秩不超过 k 的矩阵 B，设 $A+B$ 从小到大的不变因子为 $t_1, \cdots, t_n$，则满足 $s_i | t_{i+k}$.

引理

取足够大的 $M_2 = C_2 n^2 \log n, M_1 = C_1 n^2 M_2$. 则随机元素属于 $[0, M_1 - 1]$ 的规模 $n \times k$ 矩阵 $U_i (1 \le i \le 15)$ 和规模 $k \times n$ 矩阵 $V_i (1 \le i \le 15)$. 将 M_1 换成 M_2，其余用同样的方法生成 U_{16}, V_{16}.，则 $\gcd\{s_n, t_{1,n}, \cdots, t_{16,n}\} = s_{n-k}$ 的概率至少有 $1/2$. 其中 $t_{i,n}$ 表示 $A + U_i V_i$ 的最大不变因子。

- 利用上述引理，结合最大不变因子算法，就可以算出第 k 大不变因子了。

所有不变因子

引理

矩阵 A 最多有不超过 $O((n\log n)^{0.5})$ 个不同的不变因子。

所有不变因子

引理

矩阵 A 最多有不超过 $O((n\log n)^{0.5})$ 个不同的不变因子。

- 由于最小不变因子就是所有元素的 $\gcd$，最大不变因子可以开始算出来，直接套用分治算法计算一个区间的不变因子，若左端点和右端点不变因子相同，则可以直接算出区间内所有不变因子。因此，我们就得到了一个优秀的 $\tilde{O}(n^{3.5})$ 算法。

所有不变因子

引理

矩阵 A 最多有不超过 $O((n\log n)^{0.5})$ 个不同的不变因子。

- 由于最小不变因子就是所有元素的 gcd，最大不变因子可以开始算出来，直接套用分治算法计算一个区间的不变因子，若左端点和右端点不变因子相同，则可以直接算出区间内所有不变因子。因此，我们就得到了一个优秀的 $\tilde{O}(n^{3.5})$ 算法。
- ~~怎么复杂度比 $\tilde{O}(n^{\omega+1})$ 还大啊。~~ 毕竟，我们的 $\tilde{O}(n^{3.5})$ 算法相对于 $\tilde{O}(n^{\omega+1})$ 好实现得多。并且在原矩阵随机的情况下，基于不变因子的行列式算法复杂度也会降到 $\tilde{O}(n^3)$.

所有不变因子

引理

矩阵 A 最多有不超过 $O((n\log n)^{0.5})$ 个不同的不变因子。

- 由于最小不变因子就是所有元素的 $\gcd$，最大不变因子可以开始算出来，直接套用分治算法计算一个区间的不变因子，若左端点和右端点不变因子相同，则可以直接算出区间内所有不变因子。因此，我们就得到了一个优秀的 $\tilde{O}(n^{3.5})$ 算法。
- ~~怎么复杂度比 $\tilde{O}(n^{\omega+1})$ 还大啊。~~ 毕竟，我们的 $\tilde{O}(n^{3.5})$ 算法相对于 $\tilde{O}(n^{\omega+1})$ 好实现得多。并且在原矩阵随机的情况下，基于不变因子的行列式算法复杂度也会降到 $\tilde{O}(n^3)$.
- 此外，最坏情况下的复杂度也还有一些方向可以优化，一个是对于不变因子的大小分块讨论，一个是直接将线性方程组的复杂度优化成 $\tilde{O}(n^{\omega})$. 这两种方法都是可行的。

休闲小问题

- 回到原问题，现在我们已经算出了所有不变因子，但这只能够确定行列式的绝对值，如何计算行列式的正负性呢？

补充一些定义

定义 (特征多项式)

多项式 $f(A) = |\lambda I - A|$ 称为特征多项式。

补充一些定义

定义 (特征多项式)

多项式 $f(A) = |\lambda I - A|$ 称为特征多项式。

定义 (零化多项式)

满足 $f(A) = 0$ 的多项式称为零化多项式。

补充一些定义

定义 (特征多项式)

多项式 $f(A) = |\lambda I - A|$ 称为特征多项式。

定义 (零化多项式)

满足 $f(A) = 0$ 的多项式称为零化多项式。

定义 (最小多项式)

次数最小的首一零化多项式。

补充一些定义

定义 (特征多项式)

多项式 $f(A) = |\lambda I - A|$ 称为特征多项式。

定义 (零化多项式)

满足 $f(A) = 0$ 的多项式称为零化多项式。

定义 (最小多项式)

次数最小的首一零化多项式。

- 最小多项式是特征多项式的因式，特征多项式是零化多项式。

补充一些定义

定义 (特征多项式)

多项式 $f(A) = |\lambda I - A|$ 称为特征多项式。

定义 (零化多项式)

满足 $f(A) = 0$ 的多项式称为零化多项式。

定义 (最小多项式)

次数最小的首一零化多项式。

- 最小多项式是特征多项式的因式，特征多项式是零化多项式。
- 不难观察到特征多项式的常数项和行列式只差一个 $(-1)^n$，因此求矩阵行列式可以转化为求特征多项式，若最小多项式次数为 n，还可以直接求最小多项式。

钟子谦学长的论文

1.3.1 求出一个数列的最短线性递推式

我们先考虑有限数列的情况，即我们要对一个有限数列求出最短线性递推式。假设运算均在一个域上进行。

一种简单的做法是使用高斯消元消元出最短线性递推式，对于长度为 n 的有限数列复杂度为 $O(n^3)$，而使用 Berlekamp-Massey 算法可以将时间复杂度降到 $O(n^2)$。

1.4.2 求矩阵的最小多项式

$n \times n$ 的矩阵 M 的最小多项式是次数最小的使得 $f(M) = 0$ 的多项式 f。

类似定理1.2的证明，考虑 $\{I, M, M^2 \cdots\}$ 的线性递推式 $\{r_0, r_1, r_2 \cdots r_m\}$，那么我们有 $\sum_{i=0}^{m} r_{m-i} M^i = 0$，所以 $f(x) = \sum_{i=0}^{m} r_{m-i} x^i$ 即为矩阵 M 的一个零化多项式。所以矩阵 M 的最小多项式就对应着 $\{I, M, M^2 \cdots\}$ 的最短线性递推式，而由定理1.2它的阶数不超过 n，我们使用上一节中的方法求出最短线性递推式即可。

具体地，对于 n 维随机向量 u, v 我们需要求出 $\{uv, uMv, uM^2v \cdots uM^{2n}v\}$。注意到 $uM^{i+1} = (uM^i)M$，而向量乘上矩阵是可以在 $O(n^2)$ 时间内计算的，所以我们可以在 $O(n^3)$ 时间内从前往后递推出 $\{u, uM, uM^2 \cdots uM^{2n}\}$，接下来再把每一项乘上 v 即可。时间复杂度 $O(n^3)$。

如果 M 是稀疏矩阵，假设其中有 e 个非零位置，那么向量乘上 M 的结果就可以在 $O(n+e)$ 的时间内计算，我们就可以在 $O(n(n+e))$ 的时间内求出 $\{uv, uMv, uM^2v \cdots uM^{2n}v\}$，

钟子谦学长的论文

1.4.5 求稀疏矩阵行列式

对于输入的 $n \times n$ 的满秩矩阵 A，我们需要求出 $\det(A)$。

注意到我们可以快速地求出稀疏矩阵的最小多项式（见节1.4.2），而当矩阵的每个特征值的几何重数均为一时最小多项式就是特征多项式。对于 n 阶矩阵，特征多项式的常数项乘上 $(-1)^n$ 即为行列式（因为行列式即全部特征值的乘积）。

由于 A 不一定满足该性质，考虑将输入矩阵乘上一个新的矩阵 B。我们取 B 为一个 $n \times n$ 的模 p 意义下的随机对角矩阵，那么我们可以证明 AB 有至少 $1 - \frac{2n^2-n}{p}$ 的概率满足该性质[1]。求出 $\det(AB)$ 后注意到 $\det(AB) = \det(A)\det(B)$，而 $\det(B)$ 就是对角线上各个元素的乘积，相除即可得到 $\det(A)$ 的值。

[1]事实上，所有特征值的代数重数均为 1 的概率至少为 $1 - \frac{2n^2-n}{p}$，证明可参见 [7] 定理 4.3

设 A 中有 e 个非零位置，该做法的时间复杂度即为 $O(n(n+e))$。

补充一些定义

定义 (半群与交换半群)

称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

补充一些定义

定义 (半群与交换半群)

称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

定义 (半群的幺元素、幺半群)

半群 $(X, \circ)$ 中的元素 e 如果满足：对任意的 $a \in X$，必有

$$a \circ e = e \circ a = a \tag{1}$$

则称 e 为半群 $(X, \circ)$ 的幺元素。有幺元素的半群称为幺半群。

补充一些定义

中国计算机学会 China Computer Federation

定义 (半群与交换半群)

称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

定义 (半群的幺元素、幺半群)

半群 $(X, \circ)$ 中的元素 e 如果满足：对任意的 $a \in X$，必有

$$a \circ e = e \circ a = a \tag{1}$$

则称 e 为半群 $(X, \circ)$ 的幺元素。有幺元素的半群称为幺半群。

- 半群中的幺元素要么不存在，要么有且仅有一个。

补充一些定义

中国计算机学会 China Computer Federation

定义 (半群与交换半群)

称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

定义 (半群的幺元素、幺半群)

半群 $(X, \circ)$ 中的元素 e 如果满足：对任意的 $a \in X$，必有

$$a \circ e = e \circ a = a \tag{1}$$

则称 e 为半群 $(X, \circ)$ 的幺元素。有幺元素的半群称为幺半群。

- 半群中的幺元素要么不存在，要么有且仅有一个。

定义 (群与交换群)

设每个元素都有逆元的幺半群是群。若运算是交换的，则称之为交换群。

补充一些定义

中国计算机学会 China Computer Federation

定义 (环与交换环)

设 X 是一个给定的集合，在其上定义了两种二元运算 $\oplus$ 和 $\odot$。代数系统 $(X, \oplus, \odot)$ 称为环，如果它满足以下条件：

(1) $(X, \oplus)$ 是一个交换群，通常称作环的加法群；

(2) $(X, \odot)$ 是一个半群，通常称作环的乘法半群；

(3) 对于任意的 $a, b, c \in X$，有

$$a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$$

$$(b \oplus c) \odot a = (b \odot a) \oplus (c \odot a)$$

特别地，如果运算 $\odot$ 也是交换的，则称这个代数系统为交换环。

通俗地说，我们假设这个代数结构有加减乘，且乘法没有逆元。

交换环上的多项式乘法

- 交换环上的多项式乘法可以做到 $O(n \log n \log \log n)$ 复杂度，明显优于朴素算法 $O(n^2)$，作为之后要讲的交换环上的行列式得基础，有必要对这个算法稍作介绍。

经典 DFT

- 回忆一下经典 DFT 算法，算法假设输入 n 阶多项式 (表示次数不超过 $n-1$，下同)，$n = s^r$。DFT 算法以 $\{a_0, \cdots, a_{n-1}\}$ 作为输入，输出序列 $\{A_0, \cdots, A_{n-1}\}$，满足 $A_i = f(\omega_n^i)$，其中 $f(x) = \sum_{i=0}^{n-1} a_i x^i, \omega_n$ 表示 n 次本原单位根，简单优化根据：

$$f(x) = \sum_{i=0}^{s-1} f_i(x^s)$$
$$f_i(x) = \sum_{j=0}^{s^{r-1}-1} a_{sj+i} x^j \tag{2}$$

经典 DFT

- 回忆一下经典 DFT 算法，算法假设输入 n 阶多项式 (表示次数不超过 $n-1$，下同)，$n=s^r$。DFT 算法以 $\{a_0,\cdots,a_{n-1}\}$ 作为输入，输出序列 $\{A_0,\cdots,A_{n-1}\}$，满足 $A_i=f(\omega_n^i)$，其中 $f(x)=\sum_{i=0}^{n-1}a_ix^i,\omega_n$ 表示 n 次本原单位根，简单优化根据：

$$f(x)=\sum_{i=0}^{s-1}f_i(x^s)$$
$$f_i(x)=\sum_{j=0}^{s^{r-1}-1}a_{sj+i}x^j \tag{2}$$

- 根据这个式子分治，归纳即得如下引理。

经典 DFT

- 回忆一下经典 DFT 算法，算法假设输入 n 阶多项式 (表示次数不超过 $n-1$，下同)，$n=s^r$。DFT 算法以 $\{a_0,\cdots,a_{n-1}\}$ 作为输入，输出序列 $\{A_0,\cdots,A_{n-1}\}$，满足 $A_i=f(\omega_n^i)$，其中 $f(x)=\sum_{i=0}^{n-1}a_ix^i$，$\omega_n$ 表示 n 次本原单位根，简单优化根据：

$$f(x)=\sum_{i=0}^{s-1}f_i(x^s)$$
$$f_i(x)=\sum_{j=0}^{s^{r-1}-1}a_{sj+i}x^j \tag{2}$$

- 根据这个式子分治，归纳即得如下引理。

引理 (1)

对于 n 阶多项式的 *DFT*，需要不超过 $r(s-1)n$ 次加减操作，以及不超过 $r(s-1)n$ 乘法操作，乘法操作均只涉及乘上 ω_n 的若干次方。

经典 FFT

- 回到多项式乘法，现在我们要计算 $\sum_{i=0}^{n-1} a_i x^i$ 和 $\sum_{i=0}^{n-1} b_i x^i$ 的乘积 $\sum_{i=0}^{2n-2} c_i x^i$，根据经典方法可以得到序列 $\{d_i\}$，满足 $d_i = n(c_i + c_{i+n})$.

经典 FFT

- 回到多项式乘法，现在我们要计算 $\sum_{i=0}^{n-1} a_i x^i$ 和 $\sum_{i=0}^{n-1} b_i x^i$ 的乘积 $\sum_{i=0}^{2n-2} c_i x^i$，根据经典方法可以得到序列 $\{d_i\}$，满足 $d_i = n(c_i + c_{i+n})$.
- 类似地，设 $\omega = \omega_{ns}$，计算 $\sum_{i=0}^{n-1} a_i w^i x^i$ 和 $\sum_{i=0}^{n-1} b_i w^i x^i$ 的乘积，根据经典方法可以得到序列 $\{e_i\}$，满足 $e_i = n(c_i + \omega_s c_{i+n})$.

中国计算机学会 China Computer Federation

- 联立两式，得到：$(1-\omega_s)nc_i = e_i - w_s d_i, (1-\omega_s)nc_{i+n} = d_i - e_i.$

- 联立两式，得到：$(1-\omega_s)nc_i = e_i - w_s d_i, (1-\omega_s)nc_{i+n} = d_i - e_i.$
- 两边同乘 $\prod_{1<i<s,(i,s)=1} (1-w_s^i)$，且有引理：

- 联立两式，得到：$(1-\omega_s)nc_i = e_i - w_s d_i, (1-\omega_s)nc_{i+n} = d_i - e_i.$
- 两边同乘 $\prod_{1<i<s,(i,s)=1} (1-w_s^i)$，且有引理：

引理 (2)

$$\prod_{1\le i<s,(i,s)=1} (1-w_s^i) = \tau_i, \text{ 其中 } \tau_i = \begin{cases} p & i = p^k, p \text{ is prime} \\ 1 & otherwise \end{cases}$$

- 联立两式，得到：$(1-\omega_s)nc_i = e_i - w_s d_i, (1-\omega_s)nc_{i+n} = d_i - e_i.$
- 两边同乘 $\prod_{1<i<s,(i,s)=1} (1-w_s^i)$，且有引理：

引理 (2)

$$\prod_{1\le i<s,(i,s)=1} (1-w_s^i) = \tau_i, \text{ 其中 } \tau_i = \begin{cases} p & i = p^k, p \text{ is prime} \\ 1 & otherwise \end{cases}$$

- 那么不难发现我们就可以计算 $\tau_s nc_i$ 的值了，综上，有引理：

- 联立两式，得到：$(1-\omega_s)nc_i = e_i - w_s d_i, (1-\omega_s)nc_{i+n} = d_i - e_i$.
- 两边同乘 $\prod_{1<i<s,(i,s)=1} (1-w_s^i)$，且有引理：

引理 (2)

$$\prod_{1\le i<s,(i,s)=1} (1-w_s^i) = \tau_i, \text{ 其中 } \tau_i = \begin{cases} p & i = p^k, p \text{ is prime} \\ 1 & otherwise \end{cases}$$

- 那么不难发现我们就可以计算 $\tau_s nc_i$ 的值了，综上，有引理：

引理 (3)

计算 $\tau_s nc_i$ 的值需要：*(1)* 6 次 n 阶多项式的 *DFT*,*(2)* $2n\phi(s)$ 次加减操作,*(3)* $2n\phi(s)+n$ 次乘法操作 *(*只乘 w_{ns} 的幂次*)*,*(4)* $2n$ 次元素间的乘法操作.

分圆多项式

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n) = \prod_{1\le i\le s,(i,s)=1} (x - w_n^i)$。

分圆多项式

中国计算机学会 China Computer Federation

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n)=\prod_{1\le i\le s,(i,s)=1}(x-w_n^i)$。

- 分圆多项式有一些简单的性质：

分圆多项式

中国计算机学会 China Computer Federation

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n) = \prod_{1 \le i \le s, (i,s)=1} (x - w_n^i)$。

- 分圆多项式有一些简单的性质：
  1. $\Phi(w_n) = 0$, 这意味着 $(f \mod \Phi_n)(w_n) = f(w_n)$.

分圆多项式

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n) = \prod_{1\le i\le s,(i,s)=1} (x - w_n^i)$。

- 分圆多项式有一些简单的性质：
  1. $\Phi(w_n) = 0$, 这意味着 $(f \mod \Phi_n)(w_n) = f(w_n)$.
  2. $\Phi_n(x)$ 最高次数为 $\phi(n)$.

分圆多项式

中国计算机学会 China Computer Federation

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n)=\prod_{1\le i\le s,(i,s)=1}(x-w_n^i)$。

- 分圆多项式有一些简单的性质：
  1. $\Phi(w_n)=0$, 这意味着 $(f \mod \Phi_n)(w_n)=f(w_n)$.
  2. $\Phi_n(x)$ 最高次数为 $\phi(n)$.
  3. $\prod_{1\le i\le n, i|n}\Phi_i(x)=x^n-1$.

分圆多项式

中国计算机学会 China Computer Federation

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n)=\prod_{1\le i\le s,(i,s)=1}(x-w_n^i)$。

- 分圆多项式有一些简单的性质：
  1. $\Phi(w_n)=0$, 这意味着 $(f \mod \Phi_n)(w_n)=f(w_n)$.
  2. $\Phi_n(x)$ 最高次数为 $\phi(n)$.
  3. $\prod_{1\le i\le n, i\mid n}\Phi_i(x)=x^n-1$.
  4. $\Phi_{s^q}(x)=\Phi_s(x^q)$.

分圆多项式

中国计算机学会 China Computer Federation

定义 (分圆多项式)

分圆多项式是指某个 n 次本原单位根满足的最小次数的首 1 的整系数多项式，即 $\Phi(n)=\prod_{1\le i\le s,(i,s)=1}(x-w_n^i)$。

- 分圆多项式有一些简单的性质：
  1. $\Phi(w_n)=0$, 这意味着 $(f \mod \Phi_n)(w_n)=f(w_n)$.
  2. $\Phi_n(x)$ 最高次数为 $\phi(n)$.
  3. $\prod_{1\le i\le n,i|n}\Phi_i(x)=x^n-1$.
  4. $\Phi_{s^q}(x)=\Phi_s(x^q)$.
  5. 对于任意多项式 $f(x)$，存在唯一阶数不超过 $phi(n)$ 的多项式 $g(x)$，使得 $g(\omega_n)=f(\omega_n)$.

分圆多项式

- 利用第二条性质，可得 $\prod_{2\le i\le n, i|n} \Phi_i(x) = x^{n-1} + \cdots + 1$，代入 $x = 1$，即 $\prod_{2\le i\le n, i|n} \Phi_i(1) = n$，不难证明 $\Phi_i(1) = \begin{cases} p & i = p^k, p\ is\ prime \\ 1 & otherwise \end{cases}$，就是之前没证的引理 [2]。

特殊环上的多项式 DFT

- 现在我们想计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式 DFT$(n = s^r, q > r)$。根据之前的引理，DFT 只需要加法和乘单位根的幂，因此加和乘复杂度都不超过 s^q，最后为了保证解得唯一性 (例如 $1 = -\omega_2^1$)，需要将得到的每个点值看成多项式对 $\Phi_{s^q}(x)$ 取模后再代入 ω_{s^q}，可以证明取模后的表示是唯一的。

特殊环上的多项式 DFT

- 现在我们想计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式 DFT$(n = s^r, q > r)$。根据之前的引理，DFT 只需要加法和乘单位根的幂，因此加和乘复杂度都不超过 s^q，最后为了保证解得唯一性 (例如 $1 = -\omega_2^1$)，需要将得到的每个点值看成多项式对 $\Phi_{s^q}(x)$ 取模后再代入 ω_{s^q}，可以证明取模后的表示是唯一的。
- 具体取模时，可以用加减操作代替乘法操作，由于 $\Phi_{s^q}(x)$ 的首一多项式，取模时有 $s^q - \phi(s^q)$ 步减去 $t\Phi_{s^q}(x)x^l$，每当遇到 $s- = kt$ 操作时，改为执行 k 次减法即可 (加法同理)，因此一次取模操作需要 $(s^q - \phi(s^q))\mu_{s^q} = (s^q - \phi(s^q))\mu_s$ 次加减操作，μ_s 表示 $\Phi_s(x)$ 系数绝对值的和。而由引理 [1]，DFT 过程总共需要 $r(s-1)s^{q+r}$ 次加减操作。

特殊环上的多项式 DFT

- 现在我们想计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式 DFT$(n = s^r, q > r)$。根据之前的引理，DFT 只需要加法和乘单位根的幂，因此加和乘复杂度都不超过 s^q，最后为了保证解得唯一性 (例如 $1 = -\omega_2^1$)，需要将得到的每个点值看成多项式对 $\Phi_{s^q}(x)$ 取模后再代入 ω_{s^q}，可以证明取模后的表示是唯一的。
- 具体取模时，可以用加减操作代替乘法操作，由于 $\Phi_{s^q}(x)$ 的首一多项式，取模时有 $s^q - \phi(s^q)$ 步减去 $t\Phi_{s^q}(x)x^l$，每当遇到 $s- = kt$ 操作时，改为执行 k 次减法即可 (加法同理)，因此一次取模操作需要 $(s^q - \phi(s^q))\mu_{s^q} = (s^q - \phi(s^q))\mu_s$ 次加减操作，μ_s 表示 $\Phi_s(x)$ 系数绝对值的和。而由引理 [1]，DFT 过程总共需要 $r(s-1)s^{q+r}$ 次加减操作。
- 放缩一下，可以得到：

特殊环上的多项式 DFT

- 现在我们想计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式 DFT$(n = s^r, q > r)$。根据之前的引理，DFT 只需要加法和乘单位根的幂，因此加和乘复杂度都不超过 s^q，最后为了保证解得唯一性 (例如 $1 = -\omega_2^1$)，需要将得到的每个点值看成多项式对 $\Phi_{s^q}(x)$ 取模后再代入 ω_{s^q}，可以证明取模后的表示是唯一的。
- 具体取模时，可以用加减操作代替乘法操作，由于 $\Phi_{s^q}(x)$ 的首一多项式，取模时有 $s^q - \phi(s^q)$ 步减去 $t\Phi_{s^q}(x)x^l$，每当遇到 $s- = kt$ 操作时，改为执行 k 次减法即可 (加法同理)，因此一次取模操作需要 $(s^q - \phi(s^q))\mu_{s^q} = (s^q - \phi(s^q))\mu_s$ 次加减操作，μ_s 表示 $\Phi_s(x)$ 系数绝对值的和。而由引理 [1]，DFT 过程总共需要 $r(s-1)s^{q+r}$ 次加减操作。
- 放缩一下，可以得到：

引理 (4)

计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式 *DFT* 总共需要不超过 $r(s-1)\mu_s s^{q+r}$ 次加减法操作，不需要乘法操作。

- 结合引理 [3][4]，立得

- 结合引理 [3][4]，立得

引理 (5)

计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式乘积总共需要不超过 $6rs\mu_s s^{q+r}$ 次加减法操作以及不超过 $2n$ 次 $\mathbb{Z}[\omega_{s^q}]$ 上的乘法操作。

- 结合引理 [3][4]，立得

引理 (5)

计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式乘积总共需要不超过 $6rs\mu_s s^{q+r}$ 次加减法操作以及不超过 $2n$ 次 $\mathbb{Z}[\omega_{s^q}]$ 上的乘法操作。

- 现在我们回归最开始的问题，取 $M = s^Q$. 设

$$A = \sum_{i=0}^{\phi(M)-1} = a_i \omega_M^i, B = \sum_{i=0}^{\phi(M)-1} = b_i \omega_M^i$$

，我们想要计算 $C = AB$(以相同的形式表示).

- 结合引理 [3][4]，立得

引理 (5)

计算系数是 $\mathbb{Z}[\omega_{s^q}]$ 上的 n 阶多项式乘积总共需要不超过 $6rs\mu_s s^{q+r}$ 次加减法操作以及不超过 $2n$ 次 $\mathbb{Z}[\omega_{s^q}]$ 上的乘法操作。

- 现在我们回归最开始的问题，取 $M = s^Q$. 设

$$A = \sum_{i=0}^{\phi(M)-1} = a_i \omega_M^i, B = \sum_{i=0}^{\phi(M)-1} = b_i \omega_M^i$$

，我们想要计算 $C = AB$(以相同的形式表示).

- 若 $Q \leq 2$，直接当成多项式做朴素乘法并对 $\Phi_M(x)$ 取模后代入 ω_M.

- 否则令 H 满足 $2^H+2 \le Q \le 2^{H+1}+1$. 设 $q=\lfloor Q/2 \rfloor+1, r=\lceil Q/2 \rceil-1, m=s^q, n=s^r$，那么有：$q+r=Q, q-r=1/2, r \le 1, mn=Q, 2^{H-1}+2 \le Q \le 2^H+1$.

$$A=\sum_{i=0}^{\phi(M)-1} a_i \omega_M^i = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i\right)\omega_M^j$$

$$B=\sum_{i=0}^{\phi(M)-1} b_i \omega_M^i = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i\right)\omega_M^j$$

$$a(x)=\sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i\right)x^j, b(x)=\sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i\right)x^j$$

- 否则令 H 满足 $2^H+2 \le Q \le 2^{H+1}+1$. 设 $q=\lfloor Q/2 \rfloor+1, r=\lceil Q/2 \rceil-1, m=s^q, n=s^r$，那么有：$q+r=Q, q-r=1/2, r \le 1, mn=Q, 2^{H-1}+2 \le Q \le 2^H+1$.

$$A=\sum_{i=0}^{\phi(M)-1} a_i \omega_M^i = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i\right)\omega_M^j$$

$$B=\sum_{i=0}^{\phi(M)-1} b_i \omega_M^i = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i\right)\omega_M^j$$

$$a(x)=\sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i\right)x^j, b(x)=\sum_{j=0}^{n-1}\left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i\right)x^j$$

- 不难发现 $a(w_M)=A, b(w_M)=B$

- 计算 $c(x) = a(x)b(x) = \sum_{j=0}^{2n-2} (\sum_{i=0}^{\phi(m)-1} c_{in+j}\omega_m^i) x^j$

- 计算 $c(x) = a(x)b(x) = \sum_{j=0}^{2n-2}(\sum_{i=0}^{\phi(m)-1} c_{in+j}\omega_m^i)x^j$
- 那么

$$
\begin{aligned}
C &= c(\omega_M) \\
&= \sum_{j=0}^{2n-2}(\sum_{i=0}^{\phi(m)-1} c_{in+j}\omega_m^i)\omega_m^j \\
&= \sum_{j=0}^{n-1}(\sum_{i=0}^{\phi(m)-1} (c_{i,j}\omega_M^{ni+j} + c_{i,j+n}\omega_M^{ni+n+j}) \\
&= \sum_{j=0}^{n-1}(\sum_{i=0}^{\phi(m)} c_{i,j} + c_{i-1,j+n})\omega_M^{ni+j}
\end{aligned}
\tag{3}
$$

- 计算 $c(x) = a(x)b(x) = \sum_{j=0}^{2n-2} (\sum_{i=0}^{\phi(m)-1} c_{in+j}\omega_m^i)x^j$
- 那么

$$
\begin{aligned}
C &= c(\omega_M) \\
&= \sum_{j=0}^{2n-2} (\sum_{i=0}^{\phi(m)-1} c_{in+j}\omega_m^i)\omega_m^j \\
&= \sum_{j=0}^{n-1} (\sum_{i=0}^{\phi(m)-1} (c_{i,j}\omega_M^{ni+j} + c_{i,j+n}\omega_M^{ni+n+j}) \\
&= \sum_{j=0}^{n-1} (\sum_{i=0}^{\phi(m)} c_{i,j} + c_{i-1,j+n})\omega_M^{ni+j}
\end{aligned}
\tag{3}
$$

- 注意到这步转化会使 C 的阶数最多变成 $\phi(M) + n$，因此需要再次对 $\Phi_M(x)$ 取模使得 C 变为我们想到的形式。

- 通过这步转化，我们将规模为 $\phi(M)$ 的问题转化成了规模为 $\phi(m)$ 的问题，注意过程中我们会乘上常数 $\tau_s n$, 故当规模为 $M = s^Q (Q \geq 3)$ 时，会乘上常数 $k_Q = \tau_s^{H+1} s^{Q-2}$，用数学归纳法不难证明。

- 通过这步转化，我们将规模为 $\phi(M)$ 的问题转化成了规模为 $\phi(m)$ 的问题，注意过程中我们会乘上常数 $\tau_s n$, 故当规模为 $M = s^Q (Q \geq 3)$ 时，会乘上常数 $k_Q = \tau_s^{H+1} s^{Q-2}$，用数学归纳法不难证明。
- 现在我们来分析复杂度，设 α_Q, β_Q 表示加减操作和乘法操作数的复杂度，若我们取 $s \leq 3$，那么根据引理 [5]：

$$\alpha_Q \leq 6rs\mu_s s^Q + 2s^r \alpha_q + s^Q + s^r \mu_s = O(n \log n \log \log n).$$

$$\beta_Q \leq 2s^r \beta_q = O(n \log n).$$

- 若我们要计算两个 n 阶多项式相乘，只要选取 s, Q，使得 $\phi(s^Q) \geq 2n$，并将多项式的 x 替换为 w_M，用上述算法算出 C，就可以得到 $k_Q c_i$ 了 ($k_Q = s^t$，且 k_Q 级别为 $O(n \log n)$)。

- 若我们要计算两个 n 阶多项式相乘，只要选取 s, Q，使得 $\phi(s^Q) \geq 2n$，并将多项式的 x 替换为 w_M，用上述算法算出 C，就可以得到 $k_Q c_i$ 了 ($k_Q = s^t$，且 k_Q 级别为 $O(n \log n)$)。
- 为了还原答案，取两个互质的 s(取 $2, 3$ 就可以了)，设得到了 $N_1 C, N_2 C$，那么 N_1, N_2 也互质，用扩展欧几里得算法求出 $N_1 x + N_2 y = 1$ 的解 M_1, M_2，算出 $M_1(N_1 C) + M_2(N_2)C$ 即可，最终复杂度只需要再加上 $n \log n$ 次加法操作 (快速乘) 或者直接先算出代数结构中的 M_1, M_2 再进行 n 次乘法即可，都不占瓶颈。

- 若我们要计算两个 n 阶多项式相乘，只要选取 s, Q，使得 $\phi(s^Q) \geq 2n$，并将多项式的 x 替换为 w_M，用上述算法算出 C，就可以得到 $k_Q c_i$ 了 ($k_Q = s^t$，且 k_Q 级别为 $O(n \log n)$)。
- 为了还原答案，取两个互质的 s(取 $2, 3$ 就可以了)，设得到了 $N_1 C, N_2 C$，那么 N_1, N_2 也互质，用扩展欧几里得算法求出 $N_1 x + N_2 y = 1$ 的解 M_1, M_2，算出 $M_1(N_1 C) + M_2(N_2)C$ 即可，最终复杂度只需要再加上 $n \log n$ 次加法操作 (快速乘) 或者直接先算出代数结构中的 M_1, M_2 再进行 n 次乘法即可，都不占瓶颈。
- 综上，我们得到了一个需要 $O(n \log n \log \log n)$ 次加减操作，$O(n \log n)$ 次乘法操作的交换环上多项式乘积算法。

交换环上的行列式

- 在实际问题中，元素往往没有逆元，例如模合数，多项式，甚至系数模合数的多项式，传统的高斯消元或者 BM 算法都不再适用。

交换环上的行列式

- 在实际问题中，元素往往没有逆元，例如模合数，多项式，甚至系数模合数的多项式，传统的高斯消元或者 BM 算法都不再适用。
- 将行列式问题一般化，可以观察到在交换环上，行列式仍然良定义，且仍然满足之前行列式的四条性质。

交换环上的行列式

- 在实际问题中，元素往往没有逆元，例如模合数，多项式，甚至系数模合数的多项式，传统的高斯消元或者 BM 算法都不再适用。
- 将行列式问题一般化，可以观察到在交换环上，行列式仍然良定义，且仍然满足之前行列式的四条性质。
- 接下来，我们将介绍一个交换环上的 $\tilde{O}(n^{\omega/2+2})$ 算法，其中 n^{ω} 是矩阵乘法的复杂度 (现在最优算法为 $\omega = 2.373$)。

朴素的做法

- 定义新的矩阵 $B(z)=I+z(A-I)$，容易注意到 $B(0)=I, B(1)=A$。

朴素的做法

- 定义新的矩阵 $B(z)=I+z(A-I)$，容易注意到 $B(0)=I, B(1)=A$。
- 也就是说，如果计算出 $B(z)$ 的行列式，直接计算系数和就是答案了。根据定义，矩阵 $B(z)$ 的形式如下。

$$\begin{bmatrix} (a_{1,1}-1)z+1 & a_{1,2}z & a_{1,3}z & \cdots & a_{1,n}z \\ a_{2,1}z & (a_{2,2}-1)z+1 & a_{2,3}z & \cdots & a_{2,n}z \\ a_{3,1}z & a_{3,2}z & (a_{3,3}-1)z+1 & \cdots & a_{3,n}z \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1}z & a_{n,2}z & a_{n,3}z & \cdots & (a_{n,n}-1)z+1 \end{bmatrix}$$

- 考虑直接实现高斯消元法，由于矩阵只有主对角线上元素有常数项 1，在算法执行过程中，我们可以保证矩阵一直满足这个性质。因此这些元素总是有逆元，且求逆只需用到幺元、乘法和加法，符合交换环的性质，困扰我们的问题就被解决了。

- 考虑直接实现高斯消元法，由于矩阵只有主对角线上元素有常数项 1，在算法执行过程中，我们可以保证矩阵一直满足这个性质。因此这些元素总是有逆元，且求逆只需用到幺元、乘法和加法，符合交换环的性质，困扰我们的问题就被解决了。
- 观察答案形式容易注意到最终次数不超过 n，因此全过程可以对 z^{n+1} 取模，时间复杂度瓶颈在于 $O(n^3)$ 次两个交换环上的多项式乘法，朴素实现需要 $O(n^5)$ 次交换环上的运算，不管怎样，这至少是一个多项式算法。

优化

- 我们考虑将以上两个算法结合。也就是说令 $B(z) = C + z(A - C)$，且 B 的行列式容易计算。

优化

- 我们考虑将以上两个算法结合。也就是说令 $B(z) = C + z(A - C)$，且 B 的行列式容易计算。
- 一个想法是计算 B 的最小多项式，令 $\alpha_i = u^T A^i v$，若 $t_n(x)$ 是矩阵的最小多项式，则有：

$$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$

优化

- 我们考虑将以上两个算法结合。也就是说令 $B(z) = C + z(A - C)$，且 B 的行列式容易计算。
- 一个想法是计算 B 的最小多项式，令 $\alpha_i = u^T A^i v$，若 $t_n(x)$ 是矩阵的最小多项式，则有：

$$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$

- 其中 $deg(f_n(x)) < n$。

- $$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$

- $$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$
- 现在我们想要求解这个方程，注意到这是一个扩展欧几里得算法求解的问题的形式，因此我们可以通过辗转相除求解问题，考虑到需要在交换环上运算，我们需要满足以下两条性质：

- $$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$
- 现在我们想要求解这个方程，注意到这是一个扩展欧几里得算法求解的问题的形式，因此我们可以通过辗转相除求解问题，考虑到需要在交换环上运算，我们需要满足以下两条性质：
- 令 $f_{-1}(x) = x^{2n}, f_0(x) = a_0x^{2n-1} + \cdots + a_{2n-1}$，且 $f_i(x) = f_{i-2}(x) \bmod f_{i-1}(x)$，其中 a_i 是 α_i 的常数项 (不难发现 $a_i = u^TC^iv$)。则满足：
  $f_i(x) = \pm x^{2n-1-i} +$ 低次项 $(0 \le i < n)$

- $$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$$
- 现在我们想要求解这个方程，注意到这是一个扩展欧几里得算法求解的问题的形式，因此我们可以通过辗转相除求解问题，考虑到需要在交换环上运算，我们需要满足以下两条性质：
- 令 $f_{-1}(x) = x^{2n}, f_0(x) = a_0x^{2n-1} + \cdots + a_{2n-1}$，且 $f_i(x) = f_{i-2}(x) \bmod f_{i-1}(x)$，其中 a_i 是 α_i 的常数项 (不难发现 $a_i = u^TC^iv$)。则满足：$f_i(x) = \pm x^{2n-1-i} +$ 低次项 $(0 \le i < n)$
- 满足 $t_n(x) = \cdots \pm 1$，这是因为最后需要把最小多项式 (特征多项式) 的首次项系数化为 1。

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1}) = f_n(x)(\mathrm{mod}\, x^{2n-2})$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1}) = f_n(x)(\mathrm{mod}\, x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2) = f_n(x)x^2(\mathrm{mod}\, x^{2n+2})$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1}) = f_n(x)(\mathrm{mod}\, x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2) = f_n(x)x^2(\mathrm{mod}\, x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3}) = f_{n-1}(x)(\mathrm{mod}\, x^{2n-2})$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1})=f_n(x)(\mathrm{mod}\,x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2)=f_n(x)x^2(\mathrm{mod}\,x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3})=f_{n-1}(x)(\mathrm{mod}\,x^{2n-2})$
- 即 $(t_{n-1}(x))(a_0x^{2n+1}+\cdots+a_{2n-3}x^4)=f_{n-1}(x)x^4(\mathrm{mod}\,x^{2n+2})$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1})=f_n(x)(\mathrm{mod}\,x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2)=f_n(x)x^2(\mathrm{mod}\,x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3})=f_{n-1}(x)(\mathrm{mod}\,x^{2n-2})$
- 即 $(t_{n-1}(x))(a_0x^{2n+1}+\cdots+a_{2n-3}x^4)=f_{n-1}(x)x^4(\mathrm{mod}\,x^{2n+2})$
- 而我们的目标是
  $t_{n+1}(x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2+a_{2n}x+a_{2n+1})=f_{n+1}(x)(\mathrm{mod}\,x^{2n+2})$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1})=f_n(x)(\mathrm{mod}\,x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2)=f_n(x)x^2(\mathrm{mod}\,x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3})=f_{n-1}(x)(\mathrm{mod}\,x^{2n-2})$
- 即 $(t_{n-1}(x))(a_0x^{2n+1}+\cdots+a_{2n-3}x^4)=f_{n-1}(x)x^4(\mathrm{mod}\,x^{2n+2})$
- 而我们的目标是
  $t_{n+1}(x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2+a_{2n}x+a_{2n+1})=f_{n+1}(x)(\mathrm{mod}\,x^{2n+2})$
- 令 $t_{n+1}(x)=t_n(x)x-t_{n-1}(x)$，观察目标式子两边系数，不难发现总可以通过确定 a_{2n}, a_{2n+1} 的取值使得左式次数不超过 $n+1$。

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1})=f_n(x)(\mathrm{mod}\,x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2)=f_n(x)x^2(\mathrm{mod}\,x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3})=f_{n-1}(x)(\mathrm{mod}\,x^{2n-2})$
- 即 $(t_{n-1}(x))(a_0x^{2n+1}+\cdots+a_{2n-3}x^4)=f_{n-1}(x)x^4(\mathrm{mod}\,x^{2n+2})$
- 而我们的目标是
  $t_{n+1}(x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2+a_{2n}x+a_{2n+1})=f_{n+1}(x)(\mathrm{mod}\,x^{2n+2})$
- 令 $t_{n+1}(x)=t_n(x)x-t_{n-1}(x)$，观察目标式子两边系数，不难发现总可以通过确定 a_{2n}, a_{2n+1} 的取值使得左式次数不超过 $n+1$。
- 最终可以确定 $a_i=\binom{i}{\lfloor i/2\rfloor}, t_n(x)=\sum -(-1)^{(n-i+1)/2}\binom{i}{\lfloor (n+i)/2\rfloor}x^i$

- 现在我们只要找一组合法的 $a_i, t_n(x)$ 就能求解原问题了，考虑将规模 $n+1$ 的问题规约成 n。
- 考察 $t_n(x)(a_0x^{2n-1}+\cdots+a_{2n-1})=f_n(x)(\mathrm{mod}\,x^{2n-2})$
- $(t_n(x)x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2)=f_n(x)x^2(\mathrm{mod}\,x^{2n+2})$
- 还有 $(t_{n-1}(x))(a_0x^{2n-3}+\cdots+a_{2n-3})=f_{n-1}(x)(\mathrm{mod}\,x^{2n-2})$
- 即 $(t_{n-1}(x))(a_0x^{2n+1}+\cdots+a_{2n-3}x^4)=f_{n-1}(x)x^4(\mathrm{mod}\,x^{2n+2})$
- 而我们的目标是 $t_{n+1}(x)(a_0x^{2n+1}+\cdots+a_{2n-1}x^2+a_{2n}x+a_{2n+1})=f_{n+1}(x)(\mathrm{mod}\,x^{2n+2})$
- 令 $t_{n+1}(x)=t_n(x)x-t_{n-1}(x)$，观察目标式子两边系数，不难发现总可以通过确定 a_{2n}, a_{2n+1} 的取值使得左式次数不超过 $n+1$。
- 最终可以确定 $a_i=\binom{i}{\lfloor i/2\rfloor}, t_n(x)=\sum -(-1)^{(n-i+1)/2}\binom{i}{\lfloor (n+i)/2\rfloor}x^i$
- 这个取值满足性质二是显然的，用扩展欧几里得计算过程也不难证明性质一。

中国计算机学会 China Computer Federation

- 令 $c_i = -(-1)^{(n-i+1)/2}\binom{i}{\lfloor (n+i)/2 \rfloor}$，那么 $\{c_i\}$ 对应了 $\{a_i\}$ 线性递推数列。令这个数列对应的矩阵为 C，那么

$$a_i = e_1^T C^i v$$

$$e_1^T = [1, 0, 0, \cdots, 0], v = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

最后对 $\{a_i\}$ 求最小线性递推数列，即可得到矩阵 C 的行列式。

- 综上所述，现在我们可以得出我们的算法了，算法分为三步。

- 综上所述，现在我们可以得出我们的算法了，算法分为三步。
- 对于 $0 \leq i \leq 2n-1$，计算 $\alpha_i = e_1^T B^i v$

- 综上所述，现在我们可以得出我们的算法了，算法分为三步。
- 对于 $0 \leq i \leq 2n-1$，计算 $\alpha_i = e_1^T B^i v$
- 求解方程：$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$，其中 $deg(f_n(x)) < n$。

- 综上所述，现在我们可以得出我们的算法了，算法分为三步。
- 对于 $0 \leq i \leq 2n-1$，计算 $\alpha_i = e_1^T B^i v$
- 求解方程：$s_n(x)x^{2n} + t_n(x)(\alpha_{2n-1}x^{2n-1} + \cdots + \alpha_0) = f_n(x)$，其中 $deg(f_n(x)) < n$。
- 取出 $t_n(x)$ 常数项，代入 $z = 1$，稍作处理就可得到原矩阵行列式了，事实上，这个算法同时也求出了矩阵的特征多项式。

算法正确性

- 由于 $z=0$ 时，这个矩阵就是 C，保证了这个程序需要除法运算时除数的常数项永远是正负一 (除数只会是在辗转相除时的 $f_i(x)$ 的首次项以及 $t_n(x)$ 的首尾系数)，所以逆元总是存在。且由于矩阵 C 的最小多项式等于特征多项式，不难用反证法证明矩阵 $B(z)$ 的最小多项式也总是等于特征多项式。

实现

- 对于第一步，朴素实现是 $\tilde{O}(n^4)$ 的，我们可以用大步/小步算法优化。具体分为三步，开始取 r, s 保证 $rs > 2n$。

实现

- 对于第一步，朴素实现是 $\tilde{O}(n^4)$ 的，我们可以用大步/小步算法优化。具体分为三步，开始取 r, s 保证 $rs > 2n$。
- 倍增计算 $v_j(z) = B(z)^j v (0 \le j < r)$，假设当前已经算了前 $2^i - 1$ 项，那么只要计算：

$$B^{2^i} \times \left[v|Bv|\cdots|B^{2^i-1}v\right] = \left[B^{2^i}v|B^{2^i+1}v|\cdots|B^{2^{i+1}-1}v\right]$$

同时计算 $B^{2^{i+1}} = B^i \times B^i$ 即可，这样复杂度是 $\sum O(n^\omega) \cdot M(2^i) = O(n^\omega)M(r)$(朴素的循环乘法需要 $O(n^2) \cdot r \cdot M(r)$)。

- 令 $D(z) = B(z)^r$(这部分可以在上一步中顺便计算)，接着计算 $u_k^T(z) = e_1^T D(z)^k (k \le s)$，继续沿用之前做法复杂度是 $O(n^\omega) \cdot M(n)$，事实上 $n \times n$ 阶矩阵乘 $n \times s$ 阶矩阵是可以将矩阵分成 $s \times s$ 的块的。这样复杂度会降到 $O(r^2 s^\omega) \cdot M(n)$.

- 令 $D(z) = B(z)^r$(这部分可以在上一步中顺便计算)，接着计算 $u_k^T(z) = e_1^T D(z)^k (k \le s)$，继续沿用之前做法复杂度是 $O(n^\omega) \cdot M(n)$，事实上 $n \times n$ 阶矩阵乘 $n \times s$ 阶矩阵是可以将矩阵分成 $s \times s$ 的块的。这样复杂度会降到 $O(r^2 s^\omega) \cdot M(n)$.
- 或者还有一种方法，将 $u_k(z)$ 表示成：

$$u_k(z) = \sum_{l=0}^{s} u_{k,l}(z) z^r l$$

其中 $u_{k,l}(z)$ 中元素次数不超过 r.

- 令 $D(z) = B(z)^r$(这部分可以在上一步中顺便计算)，接着计算 $u_k^T(z) = e_1^T D(z)^k (k \le s)$，继续沿用之前做法复杂度是 $O(n^\omega) \cdot M(n)$，事实上 $n \times n$ 阶矩阵乘 $n \times s$ 阶矩阵是可以将矩阵分成 $s \times s$ 的块的。这样复杂度会降到 $O(r^2 s^\omega) \cdot M(n)$.
- 或者还有一种方法，将 $u_k(z)$ 表示成：

$$u_k(z) = \sum_{l=0}^{s} u_{k,l}(z) z^r l$$

其中 $u_{k,l}(z)$ 中元素次数不超过 r.

- 这样我们得到复杂度为：$s \cdot O(r^2 s^\omega) \cdot M(r)$, 两个算法复杂度一样。

- 最后对于任意 $j, k(j < r, k \leq s)$，计算 $\alpha_{kr+j}(z) = u_k^T(z)v_j(z)$.，可以看做是 $s \times n$ 的矩阵乘 $n \times r$ 的矩阵。设 $r > s$，可以将矩阵分为 $s \times s$ 的块，复杂度 $O(r^2/s \cdot O(s^\omega) \cdot M(n))$.

- 最后对于任意 $j, k(j < r, k \leq s)$, 计算 $\alpha_{kr+j}(z) = u_k^T(z)v_j(z)$.，可以看做是 $s \times n$ 的矩阵乘 $n \times r$ 的矩阵。设 $r > s$，可以将矩阵分为 $s \times s$ 的块，复杂度 $O(r^2/s \cdot O(s^\omega) \cdot M(n))$.
- 平衡各部分复杂度，取 $t = (\omega - 2)/(\omega - 1) \approx 0.273$, 这个第一部分复杂度是：$\tilde{O}(n^{3+(\omega-2)^2/(\omega-1)})$，代入 $\omega = 3/2.372863$，分别得到 $\tilde{O}(n^{3.5})/\tilde{O}(n^{3.101})$.

- 最后对于任意 $j,k(j<r,k\le s)$，计算 $\alpha_{kr+j}(z)=u_k^T(z)v_j(z).$，可以看做是 $s\times n$ 的矩阵乘 $n\times r$ 的矩阵。设 $r>s$，可以将矩阵分为 $s\times s$ 的块，复杂度 $O(r^2/s\cdot O(s^\omega)\cdot M(n))$.
- 平衡各部分复杂度，取 $t=(\omega-2)/(\omega-1)\approx 0.273$，这个第一部分复杂度是：$\tilde{O}(n^{3+(\omega-2)^2/(\omega-1)})$，代入 $\omega=3/2.372863$，分别得到 $\tilde{O}(n^{3.5})/\tilde{O}(n^{3.101})$.
- 事实上，$n\times n$ 阶矩阵乘 $n\times s$ 阶矩阵还有更优秀的算法 (采用更复杂的分块算法) 可以做到 $\tilde{O}(n^3)$ 左右，这一部分不是很有意思，咱们就不讲了。

- 最后对于任意 $j,k(j<r,k\le s)$，计算 $\alpha_{kr+j}(z)=u_k^T(z)v_j(z).$，可以看做是 $s\times n$ 的矩阵乘 $n\times r$ 的矩阵。设 $r>s$，可以将矩阵分为 $s\times s$ 的块，复杂度 $O(r^2/s\cdot O(s^\omega)\cdot M(n)).$
- 平衡各部分复杂度，取 $t=(\omega-2)/(\omega-1)\approx 0.273$，这个第一部分复杂度是：$\tilde{O}(n^{3+(\omega-2)^2/(\omega-1)})$，代入 $\omega=3/2.372863$，分别得到 $\tilde{O}(n^{3.5})/\tilde{O}(n^{3.101})$.
- 事实上，$n\times n$ 阶矩阵乘 $n\times s$ 阶矩阵还有更优秀的算法 (采用更复杂的分块算法) 可以做到 $\tilde{O}(n^3)$ 左右，这一部分不是很有意思，咱们就不讲了。
- 对于第二步，直接套用扩展欧几里得算法，朴素实现复杂度为 $\tilde{O}(n^3).$，第三步 $O(n)$，综上，我们得到一个复杂度 $\tilde{O}(n^{3+(\omega-2)^2/(\omega-1)})$ 的求解矩阵行列式与特征多项式的算法。

伴随矩阵

定义 (余子式和代数余子式)

将方阵 A 去掉 $i_1, i_2, \ldots, i_k$ 行 $j_1, j_2, \ldots, j_k$ 列的矩阵的行列式称为这 k 行 k 列的余子式，记为 $A\begin{pmatrix} i_1, i_2, \ldots, i_k \\ j_1, j_2, \ldots, j_k \end{pmatrix}$。

将 $(-1)^{i_1+i_2+\ldots+i_k+j_1+j_2+\ldots+j_k} A\begin{pmatrix} i_1, i_2, \ldots, i_k \\ j_1, j_2, \ldots, j_k \end{pmatrix}$ 称为代数余子式。

定义 (伴随矩阵)

定义 A 的伴随矩阵是一个 $n \times n$ 的矩阵，使得其第 i 行第 j 列的元素是 A 关于第 j 行第 i 列的代数余子式，记作 A^*。

伴随矩阵

引理

令 I 表示单位矩阵，那么对于任意方阵 A，满足：$AA^* = \det(A)I$。

证明.

将上式展开，得到

$$\forall 1 \le i, j \le n, \sum_{k=1}^{n} A_{i,k} \times a_{j,k}(-1)^{j+k} = [i = j] \times \det(A)$$

。

考虑将行列式按第 j 行展开的形式：$\sum_{k=1}^{n} A_{j,k} \times a_{j,k}(-1)^{j+k}$。

比较两个式子，左式结果实际相当于用第 i 行覆盖第 j 行后的行列式，因此显然等于右式。 □

域上的伴随矩阵

- 这已经是一个比较经典的问题了，2020 年的统一省选 A 卷 D2T3 就可以采用这个做法。

域上的伴随矩阵

- 这已经是一个比较经典的问题了，2020 年的统一省选 A 卷 D2T3 就可以采用这个做法。
- 分情况讨论一下原矩阵的秩即可。

交换环上的伴随矩阵

- 既然我们已经讨论了交换环上的行列式，那么能不能扩展到伴随矩阵上呢？

交换环上的伴随矩阵

- 既然我们已经讨论了交换环上的行列式，那么能不能扩展到伴随矩阵上呢？
- 回归伴随矩阵的本意，$A^*_{j,i} = \frac{\partial |A|}{\partial a_{i,j}}$，$\partial$ 表示偏微分算子。通俗地说，就是 $a_{i,j}$ 改变 da，$|A|$ 即改变 $A^*_{j,i} da$。

交换环上的伴随矩阵

- 既然我们已经讨论了交换环上的行列式，那么能不能扩展到伴随矩阵上呢？
- 回归伴随矩阵的本意，$A^*_{j,i} = \frac{\partial |A|}{\partial a_{i,j}}$，$\partial$ 表示偏微分算子。通俗地说，就是 $a_{i,j}$ 改变 da，$|A|$ 即改变 $A^*_{j,i}da$。
- 由于伴随矩阵本质上是矩阵输入元素的多项式，所以我们不必纠结这个算子的定义，只要知道他最基本的几个性质即可，就像多项式“全家桶”中的多项式求导一样，如
$\partial(f+g) = \partial f + \partial g, \partial(fg) = g\partial f + f\partial g, \frac{\partial f}{\partial h} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}.$

一般化

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n \times n}$.

一般化

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n \times n}$.
- 接着每次可以新建一个变量 v_k 并赋值：$v_k = v_i \ op \ v_j$.，最后输出的行列式答案为 v_m。

一般化

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n \times n}$.
- 接着每次可以新建一个变量 v_k 并赋值：$v_k = v_i \ op \ v_j$.，最后输出的行列式答案为 v_m。
- 现在我们想求所以初始变量的偏导数，可以反过来考虑。

一般化

中国计算机学会 China Computer Federation

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n\times n}$.
- 接着每次可以新建一个变量 v_k 并赋值：$v_k = v_i \ op \ v_j$.，最后输出的行列式答案为 v_m。
- 现在我们想求所以初始变量的偏导数，可以反过来考虑。
- 设 $f_i(v_1, \cdots, v_i)$ 表示利用前 i 个变量计算最终行列式的函数，那么初始值为：$\frac{\partial f_m}{\partial v_m} = 1, \frac{\partial f_m}{\partial v_i} = 0 (i < m)$.

一般化

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n\times n}$.
- 接着每次可以新建一个变量 v_k 并赋值：$v_k = v_i \ op \ v_j.$，最后输出的行列式答案为 v_m。
- 现在我们想求所以初始变量的偏导数，可以反过来考虑。
- 设 $f_i(v_1, \cdots, v_i)$ 表示利用前 i 个变量计算最终行列式的函数，那么初始值为：$\frac{\partial f_m}{\partial v_m} = 1, \frac{\partial f_m}{\partial v_i} = 0 (i < m).$
- 如果当前已经算好了 f_k 的所有偏导数，那么 f_{k-1} 的偏导数就是在 f_k 的基础上删除 v_k 的“贡献”并补上生成 v_k 变量的“贡献”。具体地，设 $v_k = v_i \ op \ v_j$, 则

$$\frac{\partial f_{k-1}}{\partial v_i} = \frac{\partial f_k}{\partial v_i} + \frac{\partial f_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_i}, \frac{\partial f_{k-1}}{\partial v_j} = \frac{\partial f_k}{\partial v_j} + \frac{\partial f_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_j}.$$

一般化

- 我们将计算行列式的过程抽象，假设开始有 $n \times n$ 个变量，设为 $v_1, \cdots, v_{n\times n}$.
- 接着每次可以新建一个变量 v_k 并赋值：$v_k = v_i \ op \ v_j.$，最后输出的行列式答案为 v_m。
- 现在我们想求所以初始变量的偏导数，可以反过来考虑。
- 设 $f_i(v_1, \cdots, v_i)$ 表示利用前 i 个变量计算最终行列式的函数，那么初始值为：$\frac{\partial f_m}{\partial v_m} = 1, \frac{\partial f_m}{\partial v_i} = 0 (i < m).$
- 如果当前已经算好了 f_k 的所有偏导数，那么 f_{k-1} 的偏导数就是在 f_k 的基础上删除 v_k 的“贡献”并补上生成 v_k 变量的“贡献”。具体地，设 $v_k = v_i \ op \ v_j$, 则

$$\frac{\partial f_{k-1}}{\partial v_i} = \frac{\partial f_k}{\partial v_i} + \frac{\partial f_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_i}, \frac{\partial f_{k-1}}{\partial v_j} = \frac{\partial f_k}{\partial v_j} + \frac{\partial f_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_j}.$$

- 最终答案自然就是 $\frac{\partial f_{n\times n}}{\partial v_i}$ 了。

一般化

- 这个算法是反向模式的自动微分 (The reverse mode of automatic differentiation)，扩展性很强，在机器学习等领域有着诸多应用。

一般化

- 这个算法是反向模式的自动微分 (The reverse mode of automatic differentiation)，扩展性很强，在机器学习等领域有着诸多应用。
- 利用这个算法，可以用完全和行列式相同的复杂度得到伴随矩阵。

一般化

- 这个算法是反向模式的自动微分 (The reverse mode of automatic differentiation)，扩展性很强，在机器学习等领域有着诸多应用。
- 利用这个算法，可以用完全和行列式相同的复杂度得到伴随矩阵。
- 注意到这个算法是“隐性”的，即我们并不知道具体运行方式，事实上伴随矩阵也同样有显性的复杂度一样的算法，原理和这个算法有共同之处，由于比较复杂这里就略去了，感兴趣的同学可以参考参考文献。

参考文献

[1] Eberly (Wayne), Giesbrecht (Mark), and Villard (Gilles). -On computing the determinant and Smith form of an integer matrix. In 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000), pp. 675-685. -IEEE Computing Society, Los Alamitos, CA, 2000.

[2] J. D. Dixon. Exact solution of linear equations using p-adic expansions. Numer. Math., 40:137-141, 1982.

[3] Bunch, James R.; Hopcroft, John (1974), "Triangular factorization and inversion by fast matrix multiplication", Mathematics of Computation, 28 (125): 231-236.

[4] Alman, Josh; Williams, Virginia Vassilevska (2020), "A Refined Laser Method and Faster Matrix Multiplication"

参考文献

[1] David G. Cantor and Erich Kaltofen, ON FAST MULTIPLICATION OF POLYNOMIALS OVER ARBITRARY ALGEBRAS. Acta Informatica vol. 28, nr. 7, pp. 693-701 (1991).

[2] Erich Kaltofen and Gilles Villard. ON THE COMPLEXITY OF COMPUTING DETERMINANTS, comput. complex. 13 (2004), 91 - 130.

[3] Gilles Villard. Exact computation of the determinant and of the inverse of a matrix.Workshop on Complexity —FoCM Minneapolis, Aug. 12, 2002.

[4] Gilles Villard. Kaltofen’ s division-free determinant algorithm differentiated for matrix adjoint computation. Journal of Symbolic Computation, Elsevier, 2011, 46 (7), pp.773-790.

[5] Gilles Villard. Computation of the Inverse and Determinant of a Matrix. Algorithms Seminar 2001-2002, F. Chyzak (ed.), INRIA, (2003), pp. 29-32.

中国计算机学会
China Computer Federation

预祝大家在冬令营的测试中取得满意的成绩！