从素数到合数

关于统一省选 A 卷二试第三题的探讨

绍兴市第一中学　王展鹏

2020 年 8 月

前置知识和定义

- 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。
- 余子式：将方阵 A 去掉 $i_1, i_2, \ldots, i_k$ 行 $j_1, j_2, \ldots, j_k$ 列的矩阵的行列式记为 $A \begin{pmatrix} i_1, i_2, \ldots, i_k \\ j_1, j_2, \ldots, j_k \end{pmatrix}$，如果不特别强调，默认为去掉一行一列。
- 代数余子式：即 $(-1)^{i_1+i_2+\ldots+i_k+j_1+j_2+\ldots+j_k} A \begin{pmatrix} i_1, i_2, \ldots, i_k \\ j_1, j_2, \ldots, j_k \end{pmatrix}$。
- 无向图的基尔霍夫矩阵：等于图的度数矩阵减邻接矩阵。
- 矩阵树定理：无向图的生成树个数等于其基尔霍夫矩阵的任意代数余子式。

题目描述

- 给定一个 n 个顶点 m 条边（点和边都从 1 开始编号）的无向图 G，保证图中无重边和无自环。每一条边有一个正整数边权 w_i，对于一棵 G 的生成树 T，定义 T 的价值为：T 所包含的边的边权的最大公约数乘以边权之和，即：

$$val(T) = \left(\sum_{i=1}^{n-1} w_{e_i}\right) \times \gcd(w_{e_1}, w_{e_2}, \ldots, w_{e_{n-1}})$$

  其中 $e_1, e_2, \ldots, e_{n-1}$ 为 T 包含的边的编号。
  询问 G 的所有生成树 T 的价值之和对 $P = 998244353$ 取模后的结果。
- $2 \le n \le 30, 1 \le m \le \frac{n(n-1)}{2}, 1 \le w_i \le 152501$

约定

- 在下文中，我们认为 $O(\log P)$ 是远小于 $O(n)$ 的复杂度。

朴素的做法

- 定义 S 表示所有生成树构成的集合，$W = \max(w_i)$。先做一些简单的转化：
- $$\begin{aligned}
&\sum_{T\in S}\left(\sum_{i=1}^{n-1} w_{e_i}\right) \times \gcd(w_{e_1}, w_{e_2}, \ldots, w_{e_{n-1}}) \\
=&\sum_{T\in S}\left(\sum_{i=1}^{n-1} w_{e_i}\right) \times \left(\sum_{d|\gcd(w_{e_1}, w_{e_2}, \ldots, w_{e_{n-1}})} \varphi(d)\right) \\
=&\sum_{d=1}^{W} \varphi(d) \times \left(\sum_{T\in S, d|w_{e_1}, \ldots, d|w_{e_{n-1}}} \sum_{i=1}^{n-1} w_{e_i}\right)
\end{aligned}$$
- 也就是说，我们对于所有 d，需要计算当只考虑权值是 d 倍数的边时，所有生成树边权和的和。

简单的优化

- 接下来我们考虑固定 d 以后，如何计算所有生成树边权和的和。
- 自然的想法是，枚举所有边，利用矩阵树定理计算出现次数，这样的话时间复杂度 $O(mn^3)$，不能令人满意。
- 考虑问题的组合意义，最终的生成树恰好由一条固定计算权值的边和其余 $n-2$ 条边构成，那么可以将一条边的权值看作 $w_i x+1$，计算基尔霍夫矩阵任意代数余子式的一次项系数就是问题的答案。
- 由于最后只需求一次项，全过程可以在 $\bmod x^2$ 过程下进行。高斯消元过程中，如果当前列有非零的常数项，那么可以用这行来消其余行，如果常数项全都为零，那么可以拿任意一个非零的一次项来消其余项，否则行列式就直接为 0。总的时间复杂度 $O(n^3)$。

一般的问题

- 之前都是在 $\mathrm{mod}\ 998244353$ 意义下讨论的，那能不能处理更一般的模合数情况呢？
- 我们可以直接使用类似辗转相除的方法，用一行的倍数去消另一行，分析复杂度时可以注意到每辗转相除一轮，当前最高次项系数至少除以 2 （类似于最大公约数的复杂度分析），因此总的复杂度是 $O(n^3 + n^2 \log P)$，非常优秀。

很遗憾，这是假的

- 不过很可惜，这个看似优秀的算法即使用中国剩余定理转化成 $\bmod\ p^k$ 的情况后，也存在一个致命的问题。
- 例如在 $\bmod\ 8$ 意义下，如下图所示矩阵。

$$\begin{bmatrix} 4 & a \\ x+2 & b \end{bmatrix}$$

- 可以注意到，我们很难使用 4 消掉 $x+2$。

更一般的问题

- 回到最开始的想法，我们能不能真的计算出每条边的使用次数呢？
- 假设要计算 (u, v) 的边的出现次数，考虑计算总生成树个数减去去掉这条边以后的个数，注意到减去一条边会导致基尔霍夫矩阵 $O(1)$ 个元素值变化。如下图所示矩阵。

$$\begin{bmatrix} \ddots & \vdots & \cdots & \vdots & \iddots \\ \cdots & A_{u,u} \to A_{u,u}-1 & \cdots & A_{u,v} \to A_{u,v}+1 & \cdots \\ \cdots & \vdots & \ddots & \vdots & \cdots \\ \cdots & A_{v,u} \to A_{v,u}+1 & \cdots & A_{v,v} \to A_{v,v}-1 & \cdots \\ \iddots & \cdots & \cdots & \vdots & \ddots \end{bmatrix}$$

- 因此可以简单讨论一下这四个位置的权值修改对行列式产生的影响。
- 由于同时选择 $(u, u), (v, v)$ 和 $(u, v), (v, u)$ 产生余子式相同，贡献互为相反数互相抵消。因此变化量恰好就是四个位置的代数余子式乘上对应的变化量的和。

一些 trival 的想法

- 考虑如何计算删除一行一列的余子式，我们将算法分成两部分：
  1. 枚举删除哪一行，计算剩余矩阵消元后的形式，类似下图。

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n-2} & a_{2,n-1} & a_{2,n} \\ 0 & 0 & a_{3,3} & \cdots & a_{3,n-2} & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} \end{bmatrix}$$

  2. 在枚举行的基础上，枚举删除哪一列，可以发现剩余矩阵的行列式等于一个上三角矩阵的行列式乘上一个上海森堡矩阵的行列式，也就是说，我们需要计算对于一个上海森堡矩阵任意右下角 $i \times i$ 子矩阵的行列式。

第一部分

- 考虑使用分治做法来解决这个问题。我们使用 $solve(l, r)$ 表示计算当删除的行处于区间 $[l, r]$ 内时，每行消元后的矩阵。
- 考虑每次递归到 $solve(l, mid)$ 和 $solve(mid + 1, r)$，如果递归到 $solve(mid + 1, r)$，那么我们可以将 $l \sim mid$ 这些行消成类似于上三角矩阵的形式再递归；如果递归到 $solve(l, mid)$，可以将 $l \sim mid$ 和 $mid + 1 \sim r$ 整体交换后采用类似于上面的做法解决。
- 注意到，每消一行的复杂度是 $O(n^2)$，总共会消 $O(n \log n)$ 次，因此这部分的复杂度是 $O(n^3 \log n)$。同时，辗转相除部分的复杂度为 $O(n^2 \log n \log P)$，不是瓶颈。

第二部分

- 下图就是一个上海森堡矩阵。

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ 0 & a_{3,2} & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

- 对于第二部分，相当于要求一个上海森堡矩阵任意右下角 $i \times i$ 子矩阵的行列式，这是一个经典问题。
- 令 f_i 表示右下角 $i \times i$ 子矩阵的行列式，考虑将行列式展开，可以发现一旦选择了第一行，就会变成一个子问题，因此 $f_i = a_{i,i}f_{i+1} - a_{i+1,i}a_{i,i+1}f_{i+2} + a_{i+1,i}a_{i+2,i+1}a_{i,i+2}f_{i+3} + \ldots$。
- 也就是说，我们可以在 $O(n^2)$ 的复杂度计算所有要求的行列式，要算 n 次，总的复杂度 $O(n^3)$。顺带一提，利用上海森堡矩阵可以在 $O(n^3)$ 复杂度内计算矩阵的特征多项式，由于和本次交流无关，这里略去。

举一反三

- 上述的算法能给我们一些启示，如果开始将矩阵消成较简的形式，最后的求行列式部分可能就会简单得多。
- 可以发现，在模合数意义下，我们虽然不能使用高斯消元将以一次函数为元素的矩阵消成上三角，但我们可以仅仅将常数项消成上三角。

举一反三

- 如下图

$$\begin{bmatrix} a_{1,1}x+b_{1,1} & a_{1,2}x+b_{1,2} & a_{1,3}x+b_{1,3} & \cdots & a_{1,n}x+b_{1,n} \\ a_{2,1}x & a_{2,2}x+b_{2,2} & a_{2,3}x+b_{2,3} & \cdots & a_{2,n}x+b_{2,n} \\ a_{3,1}x & a_{3,2}x & a_{3,3}x+b_{3,3} & \cdots & a_{3,n}x+b_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1}x & a_{n,2}x & a_{n,3}x & \cdots & a_{n,n}x+b_{n,n} \end{bmatrix}$$

- 我们可以枚举选择哪个一次项，贡献就是该位置的代数余子式乘上本身的一次项系数。容易注意到，对于主对角线上的位置，余子式就是主对角线上其余常数项的乘积，如果是没有常数项的位置，那么余子式就是两条对角线常数项和一个只考虑常数项的上海森堡矩阵行列式的乘积，对于右上角的位置，余子式则显然是 0。
- 容易发现，我们需要求的是任意“子区间”的上海森堡矩阵行列式。显然可以使用枚举“前缀”，计算所有“后缀”的方法得到。总的复杂度仍然是 $O(n^3)$。

小结

- 由于做初等行变换以后余子式变化无法简单地还原，所以上述算法比较难以扩展到求每条边使用次数的问题，如果有同学有扩展的想法，欢迎与我讨论。
- 因此对于这个问题，我们还是只能在 $O(n^3 \log n)$ 的复杂度内解决，接下来考虑优化这个算法。

回忆一些简单的概念

- 令 $a_{i,j}$ 表示矩阵 A 去掉第 i 行和第 j 列的余子式，那么 A 的伴随矩阵 A^* 满足 $A^*_{i,j} = (-1)^{i+j} a_{j,i}$。
- 如果原矩阵 A 可逆的话，$A^* = |A|A^{-1}$，其中 $|A|$ 表示 A 的行列式。那如果没有逆元呢？我们有如下等式

$$AA^* = |A|I$$

  其中 I 表示单位矩阵。
- 展开，得到 $\forall i, j, \sum_k A_{i,k} \times a_{j,k}(-1)^{j+k} = [i = j] \times |A|$
- 注意到，左式结果实际相当于用第 i 行覆盖第 j 行后的行列式展开后的结果，因此显然等于右式。

回到质数

- 不妨重新思考 $\mathrm{mod}\ 998244353$ 的情况，我们讨论矩阵的秩（令 n 为矩阵阶数）：
- $rank(A) = n$：矩阵有逆，可以直接根据之前的方程计算答案；
- $rank(A) \leq n - 2$：容易发现，无论如何去掉一行一列，剩余矩阵的秩都不会变大，因此任意位置的余子式一定是 0；

回到质数

- $rank(A) = n - 1$：仍然考虑 $AA^* = |A|I$ 这一等式，可以对 A 进行初等行变换以实现高斯消元。我们知道，对矩阵进行初等行变换等价于左乘初等行变换矩阵。也就是说，通过左乘一个行变换矩阵 C，使得最后等式形式变为：$A'A^* = |A|C$，其中 A' 是消元后的矩阵。
- 具体实现时，假设处理到第 k 行第 k 列，如果 $\forall i \geq k, A_{i,k} = 0$，那么说明最终必然 $A_{k,k} = 0$。由等式可以得出对于 $\forall i < k, j$，满足

$$A^*_{i,j} = \frac{0 - \sum_{i<l\leq n} A'_{i,l}A^*_{l,j}}{A'_{i,i}}$$

- 由于秩是 $n-1$，因此之后还没消过的大小为 $(n-k+1) \times (n-k)$ 矩阵上必然列满秩，容易发现 $\forall i > k, j$，满足 $A^*_{i,j} = 0$。所以最后只需暴力计算第 k 行的伴随矩阵，就可以直接递推出其余所有行了。

回到质数

- 朴素的计算一行余子式是 $O(n^4)$ 的，令人难以接受。不过容易发现，由于是同一行的余子式，就可以先删掉这一行，之后利用之前的做法，先消元以后删掉任意一列就又满足上海森堡矩阵的性质了，复杂度仍然是 $O(n^3)$。
- 但事实上，我们可以用 $A^*A = |A|I$ 类似地通过一列来推每一列，由于本题矩阵的对称性，并不需要再进行一次高斯消元，因为一行推其他行和一列推其他列的系数显然是相同的。因此，我们只需要暴力计算一个位置的余子式就可以了，时间复杂度同样是 $O(n^3)$。

递推方法

- 可以发现，不管是素数还是合数，我们的目标都是得到 $A'A^* = |A|C$，其中 A' 是上三角矩阵，对于 $\forall i, j$，我们可以得到

$$\sum_{k \geq i} A'_{i,k} A^*_{k,j} = |A| C_{i,j}$$

- 容易发现，能不能推出 $A^*_{i,j}$，取决于 $A'_{i,i}$ 是否存在逆元。如果存在，就可以通过已经算过的位置来递推此位置，同样的，如果 $A'_{j,j}$ 存在逆元，也可以推出此位置。因此，只要算出任意 i, j 满足 $A'_{i,i}, A'_{j,j}$ 都没有逆元的位置的余子式，就可以推出所有的余子式了。我们用 s 表示这样的位置个数，自然，我们希望这样的 s 不太大。

再次从质数到合数

- 那么之前的算法能不能扩展到合数呢？
- 由于模合数意义下的运算不是整环，因此不能用传统的方法讨论矩阵的秩。
- 但我们仍然可以用高斯消元来处理 $AA^* = |A|I$，用辗转相除法来消元，如果用辗转相除法求得的 $\gcd$ 与模数不互质，因为没有逆元，我们就不能用之前的递推方法来推这一行的伴随矩阵了。为了使得结果是上三角矩阵，可以将没有逆元的列换到最后，为了方便，之后就假设 A' 满足“没有逆元的列都在最后”的性质。

再次从质数到合数

- 可以证明，对 P 质因数分解后，对于它的因子 p^k，如果按上述求得的 $\gcd \bmod p = 0$ 列的数量大于 k，那么所有要求的余子式模 $p^k = 0$。这也就是说我们就可以去掉这个因子。
- 证明可以考虑把模数当成 p，如果满足上述条件，求得矩阵的秩显然 $< n - k$，同样的，去掉一行一列不会使矩阵的秩变大，因此最后消元以后的主对角线上至少有 k 个 p 的倍数，即这个位置的余子式 $\bmod p^k = 0$。
- 也就是说，设简化过后的 $P = \prod p_i^{k_i}$，那么 $s \leq \sum k_i \leq \log_2 P$。

算法实现

- 朴素地求 s^2 个位置的余子式，时间复杂度 $O(n^3s^2)$。还是不太能令人满意。
- 为了方便讨论与实现，可以将这些位置交换到矩阵的右下角，只要求出这些位置的余子式就可以了。
- 我们可以先将左上角消成上三角形式，如下图（为了方便，令 $t = n - s$）：

$$\left[\begin{array}{cccc|ccc} a_{1,1} & a_{1,2} & \cdots & a_{1,t} & a_{1,t+1} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,t} & a_{2,t+1} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{t,t} & a_{t,t+1} & \cdots & a_{t,n} \\ \hline a_{t+1,1} & a_{t+1,2} & \cdots & a_{t+1,t} & a_{t+1,t+1} & \cdots & a_{t+1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,t} & a_{n,t+1} & \cdots & a_{n,n} \end{array}\right]$$

算法实现

- 之后枚举删除哪一行，可以将其余行消掉

$$\left[\begin{array}{cccc|ccc}
a_{1,1} & a_{1,2} & \cdots & a_{1,t} & a_{1,t+1} & \cdots & a_{1,n} \\
0 & a_{2,2} & \cdots & a_{2,t} & a_{2,t+1} & \cdots & a_{2,n} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & a_{t,t} & a_{t,t+1} & \cdots & a_{t,n} \\
\hline
0 & 0 & \cdots & 0 & a_{t+1,t+1} & \cdots & a_{t+1,n} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & a_{i-1,t+1} & \cdots & a_{i-1,n} \\
0 & 0 & \cdots & 0 & a_{i+1,t+1} & \cdots & a_{i+1,n} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & a_{n,t+1} & \cdots & a_{n,n}
\end{array}\right]$$

- 最后再枚举删除哪一列，朴素使用 s^3 的高斯消元计算右下角的行列式再乘上 $\prod_{1\le i\le t} a_{i,i}$ 就行了。总的复杂度 $O(n^3 + n^2 s^2 + s^5)$。

算法实现

- 事实上我们还能做得更好。
- 仍然采用之前 $O(n^3 \log n)$ 算法的思想，把左上角消完以后，用分治算法消最后的 s 行，最后会剩下 $s \times s$ 的类似上海森堡矩阵的矩阵，依然可以使用 DP 的方法求出每个位置的余子式，总的复杂度是 $O(n^3 + n^2 s \log s)$。
- 这样的复杂度已经几乎不比朴素求行列式的 $O(n^3)$ 复杂度劣了。至此，在任意模数下，对于求解任意矩阵的伴随矩阵的问题，我们得到了一个复杂度令人满意的做法。

其他思路

- 可以发现，到最后一步以后，问题转化为求右下角一个方阵所有位置的代数余子式。考虑给矩阵加一行，加一列，那么对于 (x, y) 位置的代数余子式，等于形如下面矩阵的行列式的相反数。
-
$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,y} & \cdots & a_{1,n-1} & a_{1,n} & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,y} & \cdots & a_{2,n-1} & a_{2,n} & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ a_{x,1} & a_{x,2} & a_{x,3} & \cdots & a_{x,y} & \cdots & a_{x,n-1} & a_{x,n} & 1 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,y} & \cdots & a_{n,n-1} & a_{n,n} & 0 \\ 0 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & 0 \end{bmatrix}$$
- 这是因为根据定义计算行列式时，每行每列需要恰好选择一个位置，那么为了最后乘积非零，显然必须得选这两个 1，剩余自然就是 (x, y) 的余子式了。

其他思路

- 由于左上角 $n \times n$ 的矩阵不随着 x, y 的变化而变化，因此可以对左上角进行高斯消元，并记录消元的结果，当固定 (x, y) 以后矩阵除了 $a_{n+1,v} = 1$ 以外，一定是一个上三角矩阵。因此只需利用高斯消元使得最后一行也满足上三角矩阵性质就可以计算行列式了。
- 由于 $v > n - s$，考虑到极端情况辗转相除每次可能会进行 $\log P$ 次，时间复杂度 $O(n^3 + s^4 \log P) = O(n^3 + \log^5 P)$。如果不结合之前的递推算法，直接计算每个位置的余子式，通过一些技巧也容易做到 $O(n^3 + n^2 \log^3 P)$，虽然时间复杂度略高，但代码难度显著降低，并且可以处理多次询问余子式之类的问题。

感谢

- 感谢陈齐治学长，周雨扬、李佳衡、孟煜皓同学为本文审稿。
- 感谢聆听，预祝大家在 WC 的测试中取得好成绩！