NOI2016 模拟赛题解

Yang Jiaqi

2015 年 11 月 17 日

目录

1 群

本题来自 hihoCoder Challenge 16 的第四题, 现场只有 jiry 一个人过了.

下面设我们选出来的群为 G.

1.1 k=1 的情况

送分的.

1.2 k=2, 3, 5 的情况

设 C_i 表示大小为 i 的循环群.

当 k 为质数的时候, 我们选出的群 G 必定同构于 C_k, 那么这样的群它是存在生成元的, 因此我们只需要知道生成元是什么就可以了.

那么生成元 g 必定包含至少一个大小为 k 的轮换, 并且只能包含大小为 1 或 k 的轮换, 因此我们可以用一个简单的 DP 来计算答案.

1.3 k=4 的情况

此时有两种情况.

1.3.1 G 同构于 C_4

这种情况和 k 为质数的情况是类似的, 不同的是我们需要注意到此时我们可能包含大小为 1, 2 或 4 的轮换.

1.3.2 G 同构于 $V = C_2 \times C_2$

这是本题最精彩的地方, 由于四元素群不仅仅只有可能同构于 C_4, 还有可能同构于一个特殊的群 V, 即 Vierergruppe, 即四元群, 因此我们在这里需要更多的讨论.

V 具有这样的一个结构, 它的四个元素我们分别称为 $1, i, j, k$, 其中满足

- $i^2 = j^2 = k^2 = 1$
- $ij = k$
- $jk = i$
- $ki = j$

这实际上跟我们的四元数的结构是类似的.

考虑到这四个元素中必定有一个是 1, 也就是恒等置换, 因此我们先把 i, j, k 当成有序的, 然后我们不妨枚举剩下的三个元素中的一个, 比如说枚举 i, 然后考虑 j 和 k 的方案数, 最后再除以群 V 的自同构数 6 就可以了.

不妨枚举 i 中大小为 2 的轮换的个数, 比如说有 a 个, 那么我们就有 $n-2a$ 个大小为 1 的轮换.

这 a 个大小为 2 的轮换在 j 和 k 中有两种组合方式.

第一种是两两组合, 比如说 i 中的两个轮换 $(a\ b)(c\ d)$ 在 j 和 k 中分别对应 $(a\ d)(b\ c)$ 和 $(a\ c)(b\ d)$, 这样的话, 它对答案的贡献就是 2.

第二种是单个组合, 比如说 i 中的 $(a\ b)$ 在 j 和 k 中分别对应 $(a\ b)$ 和 $(a)(b)$, 这样的贡献仍然是 2.

因此这一部分的每一种方案的贡献是 2, 而所有方案的总贡献就求个和就可以了.

剩下的 $n-2a$ 个大小为 1 的轮换也有两种情况.

第一种是单个组合, 比如说 i 中的 (a) 在 j 和 k 中都是 (a), 这样的贡献是 1.

第二种是两两组合, 比如说 i 中的 $(a)(b)$ 在 j 和 k 都是 $(a\ b)$, 这样的贡献也是 1.

因此这一部分的贡献是所有轮换都不大于 2 的置换个数.

最后再乘一个排列数, 枚举哪些位置是大小为 1 的轮换就可以了.

2 几何题

这道题的原题是 Codechef MGCH3D.

下面我们来看看正解要怎么打吧.

我们可以发现, 问题的关键在于对于每一个 α, β, γ, 求出有多少个点对 (a_i, b_i, c_i) 与 (a_j, b_j, c_j), 使得 $a_i - a_j = \alpha$ 且 $b_i - b_j = \beta$ 且 $c_i - c_j = \gamma$, 不妨设这样的点对的个数为 $cnt[\alpha][\beta][\gamma]$.

我们不妨构造这样的一个多项式 $A(x, y, z) = \sum_{i=1}^{n} x^{a_i} y^{b_i} z^{c_i}$, 那么

$$cnt[\alpha][\beta][\gamma] = [x^\alpha y^\beta z^\gamma] A(x, y, z) A(\frac{1}{x}, \frac{1}{y}, \frac{1}{z})$$

其中右边那个方括号表示对应项的系数.

于是我们的问题就转化为了求两个多项式的乘积的问题. 直接做卷积绝对是过不了的, 我们做卷积的常用方法是 FFT, 因此我们要想办法把这个问题转化成一个可以用 FFT 做的问题.

我们暂时先不考虑指数中有负数的问题. 不妨设 m 表示每一维的系数的最大值 (在本题中 $m = 77$). 我们可以把整个多项式的每一项, 比如说 $x^\alpha y^\beta z^\gamma$ 这一项, 看作是项 $x^{\alpha\beta\gamma_{2m}}$, 也就是说把 α, β, γ 看作是一个 $2m$ 进制的数, 这样就可以把一个三维的多项式变成一维的了.

那么指数中有负数怎么办呢? 一个直接的想法是把所有项的次数都加上 m, 这样固然是可以的, 然而, 这样的话我们就需要用 $4m$ 进制, 而 FFT 的多项式的次数就会达到 $64m^3$, 达到了两千万, 这样毫无疑问会 TLE.

通过观察可以发现, 我们卷积完之后所得到的多项式的某一项的指数在 $[-m, m]$ 之间, 因此, 这启发我们可以尝试使用 $2m$ 进制. 我们不妨只把多项式 $A(\frac{1}{x}, \frac{1}{y}, \frac{1}{z})$ 的指数加上 m, 这样我们就可以只用 $2m$ 进制了.

其实到了这里, 如果你的 FFT 常数足够小的话, 已经可以 AC 了......

不过貌似我们这里的选手的 FFT 常数都比较大, 那我就来说一说怎么把常数写小一点.

我们一般是这么写的

```
for (int m = 2; m <= N; m *= 2) {
  int gap = m / 2, layer = N / m;
  for (int i = 0; i < gap; ++i) {
    double arg = sw * i * 2.0 * M_PI / m;
    Complex o(cos(arg), sin(arg));
    for (int j = i; j < N; j += m) {
      Complex u = temp[j], v = o * temp[j + gap];
      temp[j] = u + v;
      temp[j + gap] = u - v;
    }
  }
}
```

实际上应该这么写

```
for (int m = 2; m <= N; m *= 2) {
  register int gap = m / 2;
  double arg = sw * 2.0 * PI / m;
  Complex w(cos(arg), sin(arg));
  for (int i = 0; i < N; i += m) {
    Complex o(1.0, 0.0);
    for (int j = i; j < i + gap; ++j, o *= w) {
      Complex v = o * a[j + gap];
      a[j + gap] = a[j] - v;
      a[j] += v;
    }
```

```
    }
}
```

大家可以看到这两种写法的差别: 后者访问的空间更加连续, 因此这样会快很多 (我换了打法之后 6s 立马变 2s).

然而这并不是我的初衷啊......其实我本来是要给出一种用两次 FFT 来计算 (实系数) 多项式卷积的方法的 (复系数多项式可能也可以).

注意到 FFT 可以计算**复系数**多项式的插值, 不妨假设我们现在正在计算两个多项式 $a[]$ 和 $b[]$ 的卷积.

设多项式的长度为 N. 我们不妨构造两个新的多项式 $p[], q[]$, 其中 $p[i] = a[i] + I \times b[i], q[i] = a[i] - I \times b[i]$, 其中 I 表示单位复根.

我们先求出 $p[]$ 插值之后的结果 $u[]$, 并且令 $u[N] = u[0]$. 之后我们可以发现, $v[i] = u[\bar{N} - i]$.

Proof. 不妨设 $w[i] = e^{i\frac{2\pi}{N}I}$, 并且同样地, 令 $w[N] = w[0]$. 由于 $q[i] = q[\bar{i}]$, 而 $w[i] = w[\bar{N} - i]$, 并且考虑到共轭的几何意义相当于**对称**, 那么 $v[i]$ 和 $u[N - i]$ 的关系相当于整个都关于 x 轴做了一个对称, 因此 $v[i] = u[\bar{N} - i]$. □

因此我们就可以求出 $v[]$ 了. 有了 $u[]$ 和 $v[]$ 之后, 我们就可以轻松地求出原本的多项式 $a[]$ 和 $b[]$ 插值的结果.

3 树

这道题的原题是 Codechef TREEPATH.

链和菊花的情况随便 DP 一下就可以了.

3.1 60% 的做法

我们先求出每个点 i 到根结点的权值和 $dist[i]$.

不妨设 $ans[i]$ 表示用链覆盖以 i 为根的子树的答案. 很显然, 对于点 i, 覆盖 i 的链只有两种情况.

第一种情况是 i 不是覆盖 i 的链的 LCA, 这种情况我们不计入 $ans[i]$ 中, 因为它使用了 i 的祖先结点.

第二种情况是 i 是覆盖 i 的链的 LCA, 此时, i 可能会与它的至多两个孙子联系在一起.

不妨先考虑 i 只和一个孙子 j 连在一起的较为简单的情况, 此时, 我们已经求出了一条 i 到 j 的链, 那么剩下部分的方案数是多少呢? 实际上, 这个方案数就等于我们把 i 到 j 这条链去掉之后, 剩余的 “悬挂” 部分的 ans 的乘积.

那么 i 和两个孙子 j 和 k 两个孙子连在一起的情况也是一样的.

我们可以重新 DFS 一遍, 对于 i 的每一个儿子 j, 我们计算出 i 到 j 这条路径上, 除掉 i 到 j 这条链之后, 并且不考虑点 i 的儿子, 所有连通块的 ans 的乘积, 设这个值为 $cur[i][j]$.

我们考虑我们能够选取孙子 j 和 k 的充要条件是什么, 实际上, 这只需要 $dist[j]+dist[k]-2\times dist[i]+val[i]\geq 0$, 也就是对于一个 j, 我们只需要求出所有满足 $dist[k]\geq 2\times dist[i]-val[i]-dist[j]$ 的 k 的贡献和就可以了.

下面我们设 $up[i][j]$ 表示点 j 往 i 这个方向跳, 跳到和 i 距离为 1 时所得到的点.

那么我们可以维护一棵线段树, 其中坐标为 p 的点存满足 $dist[k]=p$ 的 k 的贡献和, 一开始我们将这棵线段树清空. 之后对于 i 的每一个儿子 j, 我们枚举以 j 为根的子树中的所有点 k, 那么点 k 的贡献就是

$$cnt[i][k]\times query(2\times dist[i]-val[i]-dist[j],\infty)\times cur[i]\div ans[j]$$

理解一下这个式子: 其中第一项是 i 到 k 这条路径的贡献, 第三项是 i 的儿子的贡献, 注意到如果 k 连接的是 u, 那么我们实际上需要除掉一个 $ans[up[i][u]]$, 那么我们可以在把这个值加入线段树的时候就除掉它, 这样就不会有问题了.

做完以 j 为根的子树之后, 我们可以再用 $cnt[i][k]\div ans[j]$ 来更新到线段树中.

注意这里有个模 10^9+7 的问题, 我们可能是求不到逆元的 (我没有特意去卡这个, 因为我感觉比较难构造). 一种解决的方法是你自己写一个整数类型, 把一个数表示成 $a+b\times(10^9+7)$ 的形式.

另外一种方法是维护前缀积和后缀积, 我们将主要采用这种做法, 这样大概需要增加一个把整棵线段树都乘上一个值的操作, 这是很容易实现的.

这样时间复杂度 $O(n^2\log n)$.

3.2 100% 的做法

设点 i 的父亲为 $fa[i]$, 以 i 它为根的子树的结点数为 $size[i]$, i 的儿子中 $size$ 最大的为 $mx[i]$.

我们把每个点的线段树可持久化一下, 设点 i 线段树为 $tree[i]$.

我们可以直接令 $tree[i]=tree[mx[i]]$, 然后我们再遍历 i 的子树中的剩余结点, 并更新线段树.

这里相当于应用了启发式合并的思想, 因此总的时间复杂度就变成了 $O(n\log^2 n)$.