绍兴一中省选模拟赛

解题报告

By Yuekai Jia

2014 年 3 月 8 日

同分异构体

【简要题意】

求 N 个点，每个点度数不过超过 4 的无标号有根树($T = 2$)、无根树($T = 1$)，和包含一个长度为 M 的环，其余每个点连出的子树大小不超过 N 的无标号环加外向树个数($T = 3$)。

【算法要点】

DP，组合数学，Pólya 计数法，数论

【算法一】

先考虑有根树。

设 $f[i]$ 表示 i 个点的满足要求的有根树个数，即三叉树。枚举前两棵子树的大小(j, k)，第三棵子树大小也可以随之算出($l = i - j - k$)，同时还要满足 j, k, l 单调不减。先假设 j, k, l 互不同，则对 $f[i]$ 的贡献就是 $f[j]f[k]f[l]$；如果三个数都相同，则答案可以看做从 $f[j]$ 种不同的子树中选出三种进行组合，且选出的子树可以相同，即可重组合问题。于是对 $f[i]$ 的贡献就是 $\binom{f[j]+1}{2}$；同理，如果两个数相同，不妨设 $j = k \neq l$，对 $f[i]$ 的贡献为 $\binom{f[j]+2}{3} \cdot f[l]$。

时间复杂度 $O(N^3)$，期望得分 $15 \sim 25$ 分。

【算法二】（标准算法 1）

考虑对算法一的优化。

上述算法的瓶颈在于要枚举每棵子树的大小，且如果问题扩展到 K 叉树该算法就无能为力了。不枚举子树大小的做法也非常显然，即用类似背包的方法优化。设 $g[i][j]$ 表示当前做了第 i 棵子树，前 i 棵子树的大小和为

j 的方案数，则 $f[i]=g[3][i-1]$。然后考虑用 $f[i]$ 更新 $g[j][k]$，枚举大小为 i 的子树的个数 l，则可以用 $\binom{f[i]+l-1}{l}\cdot g[j][k]$ 更新 $g[j+l][k+i\cdot l]$。

时间复杂度 $O(N^2)$，期望得分 30 分。

【算法三】（标准算法 2）

现在考虑无根树。设 $f'[i]$ 表示 i 个点的满足要求的无根树个数。

考虑树中的一些特殊点，当以这些点为根建树时，每棵子树的大小不超过 $\frac{N}{2}$。这样的点对任何树都是存在的，不妨定义其为树的“重心”。

当 N 为奇数时，可以证明，树的重心有且仅有 1 个。以这个点为根计数时，只要满足每棵子树的大小不超过 $\frac{N}{2}$，树的点数和为 N，则对于任意形态的子树，它们连起来都是不同的树。这时的答案即选 4 棵大小不超过 $\frac{N}{2}$ 的子树的方案数，可以用算法二中的 $g[4][i]$ 更新 $f'[2i+1]$。

当 N 为偶数时，可以证明，树的重心最多只有 2 个，且如果有两个重心，则这两点之间有边相连，这条边把整棵树分成的两部分大小都为 $\frac{N}{2}$。当树有两个重心时直接套用上述公式可能会导致重复计数，即一棵树在以一个重心为根时被算了一遍，在以另一个重心为根时又被算了一遍。不过有一种情况不会被多算：分别以两个重心为根时子树都一模一样，即重心之间的边把整棵树分成的两部分一模一样。所以多算的方案数就是 $\binom{f[\frac{N}{2}]}{2}$，可以用算法二中的 $g[4][2i-1]-\binom{f[i]}{2}$ 更新 $f'[2i]$。

时间复杂度 $O(N^2)$，期望得分 30 分。

结合算法二，期望得分 60 分。

【算法四】（标准算法 3）

考虑第三类数据。

首先求出各种大小的有根树的个数和，记为 K。则问题可以转化为，在一个长度为 M 的环的每个点涂上 K 种颜色，求旋转、翻转后本质不同的方案数。显然用 Pólya 计数法做，共有 $2M$ 种置换，M 种旋转置换的循环节数分别为 $\gcd(i,M)$；如果 M 为奇数，每种置换都有 1 个长度为 1 的循

环节和 $\frac{M-1}{2}$ 个长度为 2 的循环节，共 $\frac{M+1}{2}$ 个循环节；如果 M 为偶数，有 $\frac{M}{2}$ 种置换有 $\frac{M}{2}$ 个长度为 2 的循环节，有 $\frac{M}{2}$ 种置换有 2 个长度为 1 的循环节和 $\frac{M}{2}-1$ 个长度为 2 的循环节，共 $\frac{M}{2}+1$ 个循环节。所以最后的答案为

$$\sum_{i=1}^{M} K^{\gcd(i,M)} + \begin{cases} M \cdot K^{\frac{M+1}{2}} & M \text{ 为奇数；} \\ \frac{(K+1)M}{2} \cdot K^{\frac{M}{2}} & M \text{ 为偶数。} \end{cases}$$

化简前面那个式子为

$$\sum_{d|M} \varphi\left(\frac{M}{d}\right) K^d$$

求出 M 的所有约数，然后根据

$$\sum_{d|n} \varphi(d) = n$$

即可递推求出每个约数的 φ。

时间复杂度 $O(N^2 + \sigma(M)^2 \log_2 M)$，期望得分 40 分。

结合算法二、三，期望得分 100 分。

本题的 M 还可以开大到 10^{18}，用 Pollard rho 做，但由于需要考虑的地方比较多(如 $P|M$)，实在太胖了，就不出了。

三维变换

【简要题意】

三维空间中 N 个点，支持区间平移、缩放、旋转，每次询问点的坐标、区间内相邻点的距离和。

【算法要点】

计算几何，矩阵，数据结构

【第一部分】

直接讲标准算法，标准算法分为两部分。

首先这三种变换都是线性的，一个点做变换可以表示成一个向量与矩阵的乘积。如，把点 (x, y, z) 表示成一个 1×4 的向量 $\begin{bmatrix} x & y & z & 1 \end{bmatrix}$，一个平移变换 (a, b, c) 可以表示成一个 4×4 的矩阵

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

则点平移后的向量即它们的乘积。

现在假设对每种变换都求出了它们对应的矩阵(求法见第二部分)，如何维护每个点的坐标和相邻点的距离和。我们维护每个点被乘上的矩阵的积，于是问题就变成了区间修改、单点询问，直接套线段树等数据结构即可。

树状数组可不可以呢？由于是区间修改、单点询问，显然可以差分后维护前缀和。但是由于是维护矩阵的积，而矩阵乘法不满足交换律，所以就不能用树状数组，乖乖写线段树吧！

然后是维护区间内相邻点的距离和。记 $l[i] = |P_iP_{i+1}|$。可以发现，平移、旋转变换不会改变区间内部的信息，即 $l[L] \sim l[R-1]$ 都不变，只有 $l[L-1], l[R]$ 会变；对于缩放变换，也只是把 $l[L] \sim l[R-1]$ 都乘上了 k。所以问题就变成了区间修改、区间询问，再套一棵线段树等数据结构即可。

【第二部分】

现在考虑如何求出三种变换对应的矩阵。

平移矩阵比较简单。对于一个平移变换 (a,b,c)，它对应的矩阵

$$Trans(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

然后是缩放变换。对于一个缩放变换 (a,b,c,k)，可以看成先把点平移到原点，再做缩放变换，最后再移回来。于是它对应的矩阵

$$Scale(a,b,c,k) = Trans(-a,-b,-c) \begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Trans(a,b,c)$$

$$= \begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ (1-k)a & (1-k)b & (1-k)c & 1 \end{bmatrix}$$

最后是旋转变换。先考虑每次绕坐标轴旋转。根据平面上的旋转公式，易得

$$Rotate_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotate_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotate_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

对于一个普通的旋转变换 $(a, b, c, a', b', c', \theta)$[1]，基本思路是，首先把点$(a, b, c)$平移到原点，再把旋转轴分别绕 x 轴、 y 轴旋转合适的角度后与 z 轴重合，然后绕 z 轴旋转 θ 角，最后再转回来、移回来。

1. 把点(a, b, c)平移到原点。即乘上一个矩阵

$$T = Trans(-a, -b, -c) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}$$

2. 把旋转轴绕 x 轴旋转合适的角度，使之与 xOz 平面重合。令点 $P = (a', b', c')$，旋转的角度 α 即 OP 在 yOz 平面上的投影与 z 轴的夹角，令 $u = \sqrt{b'^2 + c'^2}$，则

$$\cos\alpha = \frac{c'}{u} \quad \sin\alpha = \frac{b'}{u}$$

$$R_x = Rotate_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c'}{u} & \frac{b'}{u} & 0 \\ 0 & -\frac{b'}{u} & \frac{c'}{u} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[1]为了方便，这里假设方向向量 (a', b', c') 为单位向量，即 $a'^2 + b'^2 + c'^2 = 1$。

3. 把已在 xOz 平面内的旋转轴绕 y 轴旋转合适的角度，使之与 z 轴重合。当前的 P 点坐标为 $(a', 0, u)$，旋转的角度 β 即 OP 与 z 轴的夹角，但现在变成了顺时针，所以要取负号。令 $v = \sqrt{a'^2 + u^2} = \sqrt{a'^2 + b'^2 + c'^2} = 1$，则

$$\cos\beta = \frac{u}{v} = u \quad \sin\beta = \frac{a'}{u} = a'$$

$$R_y = Rotate_y(-\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u & 0 & a' & 0 \\ 0 & 1 & 0 & 0 \\ -a' & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. 绕 z 轴旋转 θ 角。这个就不用说了，

$$R_z = Rotate_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5. 转回来、移回来。即上述三个矩阵 T, R_x, R_y 的逆矩阵。不过由于变换矩阵的特殊性，可以很容易求出

$$T^{-1} = Trans(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

$$R_x^{-1} = Rotate_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c'}{u} & -\frac{b'}{u} & 0 \\ 0 & \frac{b'}{u} & \frac{c'}{u} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y^{-1} = Rotate_y(\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u & 0 & -a' & 0 \\ 0 & 1 & 0 & 0 \\ a' & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

最后我们终于得出了旋转变换 $(a, b, c, a', b', c', \theta)$ 对应的矩阵

$$Rotate(a, b, c, a', b', c', \theta) = TR_xR_yR_zR_y^{-1}R_x^{-1}T^{-1}$$

这么大一坨东西看着非常不爽，化简一下，可以得到一个比较好看的式子

$$T \begin{bmatrix} a'^2(1-\cos\theta)+\cos\theta & a'b'(1-\cos\theta)+c'\sin\theta & a'c'(1-\cos\theta)-b'\sin\theta & 0 \\ a'b'(1-\cos\theta)-c'\sin\theta & b'^2(1-\cos\theta)+\cos\theta & b'c'(1-\cos\theta)+a'\sin\theta & 0 \\ a'c'(1-\cos\theta)+b'\sin\theta & b'c'(1-\cos\theta)-a'\sin\theta & c'^2(1-\cos\theta)+\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T^{-1}$$

至此，原问题得到解决。

时间复杂度 $O(4^3M\log_2 N)$，期望得分 100 分。

语言识别

【简要题意】

给出 15 种不同风格的语言(含有奇怪的语言)，对输入的代码判断它是什么语言。提交答案题。

【算法要点】

非传统题，乱搞

【数据特点】

数据编号	代码个数	包含语言	备注
1	47	全部	送分点(47个样例)。
2	200	全部	手算点。
3	2000	C++、Pascal、Java、Python	各语言特点鲜明。
4	2000	Perl、PHP、Haskell、Fortran	
5	2000	BG、Brainfuck、LaTeX、Text	全是奇怪的语言。
6	2000	HTML、XML、PHP、Text	
7	2000	Perl、Haskell、Python、Ruby、Fortran、Text	各语言不易区别。
8	2000	全部	
9	2000	全部	
10	2000	全部	

【算法一】

首先可以根据不同语言的特点大致地把它们分成几类：

1. 类 C++ 语言：C++、Java、PHP、Perl；

2. 类 Pascal 语言：Pascal、Fortran；
3. 类 HTML 语言：HTML、XML；
4. 类 Python 语言：Python、Ruby、Haskell；
5. 奇怪的语言：BG(自己造的)、Brainfuck(代码只包含 8 种字符)、LaTeX(文档排版语言)、Text(纯文本，可能包含各种干扰的语句)。

然后根据不同类别语言的差异就可以区分出一部分了。

【算法二】

观察发现，有一些语言有自己独一无二的特点，如：C++ 的 `#include`，PHP 的 `<?php?>`，XML 的 `<?xml version="1.0">` 等等。于是大部分语言就可以这样被轻松识别。

【算法三】

上述算法都有一个致命缺陷：对包含各种干扰的语句的纯文本无能为力。于是现在的问题就是如何高效识别纯文本与正常代码。首先显然有一点，对于正常代码，其中的运算符一定特别多，所以可以根据文本中运算符占的比例区别正常代码与纯属文本。

【算法四】

由于纯文本也可能包含很多运算符，上述算法三也不一定有效。对于这种纯文本，它们显然是杂乱无章、参差不齐的，所以可以自行定义一段文本的“混乱程度”来判断是纯文本还是正常代码。

【算法五】

一般情况下正常代码都有代码风格，如缩进、对齐等，这也可以用来区分。

【算法六】

正常代码各括号都会匹配，而随机的纯文本就不一定了。

【算法七】

正常代码中单词的种类比较少，也就几个关键字和变量、函数名，而纯文本中单词的种类会比较多。

【算法八】

直接判长度，一般来自 OJ 提交记录中的代码很少有达到 5、6K 的吧？

【算法 N】

请当场得分最高的同学来讲……