2024「钉耙编程」中国大学生算法设计超级联赛（6）题解

A. 造花（简单版）

要能把一个树分成两个菊花图，要删去的点的度数必定为 2。

先考虑一个简单的做法，检查所有的度数为 2 的点，依次检查删去他们之后，剩余的图是不是菊花图，检查一次的复杂度为 $O(n)$ 的话，这样的方法复杂度是 $O(n^2)$ 的，无法接受。

考虑是否需要检查所有的度数为 2 的点。反推回来，如果两个菊花图被一个点连接了起来，原图度数不为 1 的点不应该有太多，仔细思考可以发现应该不超过 5 个，所以每张图只需要检查五个不同的度数为 2 的点就好了（当然多检查几个也无妨），这样整个的复杂度即为 $O(n)$，可以通过。

检查整个图所有极大连通子图是不是菊花图，可以考虑使用并查集，对每个连通块记录点的个数，边的个数和度数为 1 的点的个数，对于一个连通块，如果边的个数比点的个数少一个，且度数为 1 的点的个数和连通块的点的个数差值不大于 1，就说明这个子图是菊花图。

B. 造花（困难版）

简单版的思路显然不能直接照搬，因为原图度数不为 1 的点可以有很多。但是依然可以考虑一个问题：真的需要每个点都检查一下是否是混沌点吗?

首先可以考虑一条边上的两点 u 和 v，如果存在点 x 和 u 相连（x 和 v 不同），且存在点 y 和 v 相连（y 和 u 不同，但 x 和 y 可以相同），则上面提到的这四个点必须至少删除一个，否则就会形成长度至少为 3 的链或一个环，就与所有子图都是菊花图违背。

因此，考虑枚举所有的边，对于每条边 (u,v)，枚举这两个点的所有相邻点以寻找点 x 和 y。如果存在这样的四个点满足上述关系，则依次检查它们是否是混沌点（检查方法同简单版），此时其它点必定不是混沌点，无需检查。否则如果不存在这样的四个点，则原图一定已经满足「所有极大连通子图都是菊花图」的要求（因为整个图必定无环且最长链长度小于等于 2），此时所有点都是混沌点。

最终复杂度和简单版一样为 $O(n)$。

C. 飞车狂飙

题目描述

有一条长为 n 的轨道，在平面上依次铺设，问：

- 轨道是否自交
- 轨道是否成环

数据范围：$1 \le n \le 10^5$

题解

初始方向对最终答案没有影响，因此可以任意取初始朝向后模拟这个过程即可。使用 `set` 维护已经铺设过的格子坐标即可判断轨道是否自交。结束后除了判断是否回到原点外，还需判断当前朝向与初始朝向是否相同，否则将在判环上出现问题。

时间复杂度：$\mathcal{O}(n \log n)$。

D. 不醒人室

题目描述

n 节课，m 个小憩时间段。每次小憩可以保持接下来二倍时间不困，但超过时间就会困。问：

- 会不会在某节课上睡觉
- 会不会在某节课上很困

数据范围：$1 \leq n, m \leq 10^5$

题解

首先将上课时段和小憩时段标记后合在一起排序，这个部分可以简单使用排序函数，也可以直接归并（两部分时段均是有序的）。

首先检查会不会在课上睡觉的情况，由于上课时间段互不相交，小憩时间段也互不相交，因此这一部分仅需检查合并排序后的时间段是否存在相交情况，如果有相交，则说明会在课上睡觉。

接着检查会不会在某节课上很困，可以考虑在小憩之后维护不困时间段，对于上课时段检查是否完全在不困时段内。如果存在上课时段不完全在不困时段内，则说明会在这节课上很困。

这两部分的检查时间复杂度均为线性。

时间复杂度：$\mathcal{O}((n+m)\log(n+m))$ 或 $\mathcal{O}(n+m)$。

E. 交通管控

题目描述

有 k 个路口，n 个操作杆。这 n 个操作杆只有使用和不使用两种状态。使用一个操作杆，可以对 k 个路口的红绿灯产生影响，不使用就不会产生影响。目前所有路口都是绿灯，问对于所有 k 个路口红绿灯的情况，有多少种使用操作杆的方法能达成这样的状态。对 M 取模。

数据范围：$1 \leq n \leq 500, 1 \leq k \leq 10$

题解

由于操作杆对红绿灯的影响互相独立，考虑对所有红绿灯状态进行状压 DP。记 $f_{i,j}$ 为使用前 i 个操作杆，使得红绿灯状态为 j 的方案数，则转移为

$$f_{i,j} = f_{i-1,j} + \sum_{k \in \mathcal{U}, \mathrm{stat}(k,s)=j} f_{i-1,k}$$

其中 $\mathcal{U}$ 表示所有红绿灯的状态，$\mathrm{stat}(k,s)$ 表示状态 k 经过操作杆 s 变换后的状态。

最终答案即为 $f_{n,\cdot}$。

实现上需要注意：

1. 可以使用滚动数组优化空间复杂度，同时会带来时间收益
2. 在模意义下，不能简单判断 $f_{n,\cdot}$ 是否为 0 来判断此状态是否存在，因为 $f_{n,\cdot}$ 在不取模的情况下可能为模数的整数倍。需要附加一个状态记录状态是否存在，或者为模意义下计算添加「是否是非模意义下 0」的标志位

时间复杂度：$\mathcal{O}(nk3^k)$。

F. 解方程

对方程左右模 q，则 $x^2 = -py^2 \pmod q$，因此 $-p$ 是模 q 的二次剩余，设 $e^2 = -p \pmod q$。

将方程看成 $(x + \sqrt{-p}y)(x - \sqrt{-p}y) = q$。

因此若方程有解，设 $x - ey = 0 \pmod q$。

则 $x - ey = kq \to ey - kq = x$。

注意到 x, y 均小于 $\sqrt{q}$。

因此格：

$$\begin{pmatrix} 1 & e \\ 0 & q \end{pmatrix}$$

存在短向量 $\vec{v} = (y, x)$，$|\vec{v}| = O(\sqrt{q})$，并且我们可以证明该向量为格中的最短向量。

类似于求 gcd 的欧几里得算法（也是 LLL 算法的 2×2 情况），我们可以对这个格进行规约，复杂度 $O(\log q)$ 一组。

具体算法流程可以参考 Lattice reduction - Wikipedia。

G. 树上 MEX 问题

题目描述

给定一棵 n 个点的树，每个点有点权，点权是 0 到 $n-1$ 的排列，求所有连通块（连通导出子图）的 MEX 之和。

数据范围：$1 \le n \le 10^5$

题解

记点权为 i 的点编号为 $node_i$。

考虑使用公式 $E(X) = \sum_{i=1}^{\infty} P(X \ge i)$，把问题转化为：对于 $i = 1, 2, 3, \ldots, n$ 求 $MEX \ge i$ 的连通块数量。$MEX \ge i$ 的连通块即包含 $node_0, node_1, node_2, \ldots, node_{i-1}$ 的连通块。

首先以 $node_0$ 为根，进行树形动态规划，对于点 u 求出在 u 的子树内包含点 u 的的连通块数量 dp_i，转移为 $dp_u = \prod_{v \in son_u} (1 + dp_v)$。

对于一个确定的 i，注意到所有 $MEX \ge i$ 的连通块都至少包含了一个连通点集，$node_0$ 是这个连通点集的根，记这个连通点集为 S_i。$MEX \ge i$ 的连通块数量即所有不在 S_i 中但和 S_i 内的点有直接连边的点 v 的 $dp_v + 1$ 的乘积 num_i。

考虑如何从 S_i 和 num_i 得到 S_{i+1} 和 num_{i+1}：若 $node_i$ 在 S_i 中，则 $S_{i+1} = S_i$，$num_{i+1} = num_i$；否则从 $node_i$ 开始，不断跳父边，同时将沿途的点加入 S_i 中最终得到 S_{i+1}，并在 num_i 的基础上去掉和新增一些点的 dp 值贡献即可得到 num_{i+1}。

注意到每个点最多只能加入到 S 中一次，每个点的贡献最多只会加入 num 中和从 num 中去掉一次，因此时间复杂度为 $O(n \log \mathrm{mod})$，其中 $\log \mathrm{mod}$ 为从 num 中去掉贡献时计算逆元的复杂度。

可能出现 $dp_v + 1$ 在模意义下为 0 的情况，此时需要另外维护 dp 和 num 中的 0 的个数而不能求逆元，由于出题人疏忽没有考虑到这种情况，故测试数据中没有特殊构造这种情况。

H. 树形 DNA

首先由于 S 和 T 的两棵子树是不等价的，所以我们可以将其视为字符集大小为 2 的 Trie 树。且由于 T 的叶子节点数量不超过 20，则我们可以使用不超过 20 个 01 串来构建出 T。

不妨设 T 是由 $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k$ 这几个 01 串插入 Trie 树中构建而成的，其中 $k \leq 20$。对于 S 中的某个节点 u，只需检查 u 对应的子树在看做 Trie 后是否包含这 k 个串即可。

至于具体实现，首先我们定义 $\mathsf{anc}(v, a)$ 表示 v 节点的 a 级祖先。在 S 中每个节点开一个变量 cnt_v。对 S 进行 dfs，在搜索到节点 v 时，枚举 $\mathbf{t}_i \in \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k\}$，判断 $\mathbf{t}_i$ 是否是在 S 中从根到 v 节点路径组成字符串的后缀，如果是，则令 $\mathsf{cnt}[\mathsf{anc}(v, |\mathbf{t}_i|)] := \mathsf{cnt}[\mathsf{anc}(v, |\mathbf{t}_i|)] + 1$。最后回溯时判断 $\mathsf{cnt}[\mathsf{anc}(v, |\mathbf{t}_i|)] = k$ 是否成立即可。可以使用哈希或者 ACAM 优化匹配过程。时间复杂度 $\mathrm{O}(m + nk)$。

可能有用的：

- 在验题时发现有的队伍是将 T 的虚树建出来加速匹配的（T 的虚树大小不超过 40）。这种做法可能代码实现难度比较高且常数较大，不过时限给的很松，也应该能通过。
- 关于为什么要将空指针设为指向根（而不是 -1 之类的）：最早提出的做法是基于 ACAM 的，而 ACAM 有一种初始时将所有指针指向根的 trick，为了方便写 ACAM 便将数据写成了这样。

I. 数字加减

题目描述

一个初始为 0 的数字，每个时刻可以选择加一减一或者不变。n 个限制，t_i 时刻末数字不能在 $[l_i, r_i]$ 范围内；q 次询问，t'_i 时刻末数字有多少可能取值（只考虑 t'_i 时刻末之前的限制）。

数据范围：$1 \leq n, q \leq 5 \times 10^5$，$-10^9 \leq l_i \leq r_i \leq 10^9$，$1 \leq t_i, t'_i \leq 10^9$

题解

可以把每个时刻数字的可能的取值想象成数轴上的若干个不相交的线段，每过一秒每个线段左端点减一，右端点加一；每次询问的答案即为询问时刻末所有线段的长度之和。

随着时间增加，相邻的线段会在某个确定的时刻末合并；

每个限制等价于删除若干个线段，或者使得某些线段变得更短；

于是我们可以通过按照时间顺序依次执行「线段合并」和「线段删除」这两种操作来维护数轴上所有线段及长度之和。

实现过程中一些可能需要考虑清楚的细节：

- 并不需要实时更新线段在某一时刻具体的长度。对于每个线段其长度都可以通过一个中心点、0 时刻线段的半径以及当前时刻来计算；
- 对于受「线段删除」操作影响的所有线段，首先要撤销这些线段与相邻的左侧线段和右侧线段发生合并的操作，然后再给区间删除后形成的新线段添加区间合并操作。

由于每次「线段删除」操作最多只能增加一个线段，故「线段删除」操作总共最多只会删除/修改 $\mathcal{O}(n)$ 条线段，同理「线段合并」操作的执行次数也是 $\mathcal{O}(n)$ 级别的；

可以用 set 分别维护两种操作以及询问按照时间先后的顺序、线段在数轴上的前后顺序。

时间复杂度：$\mathcal{O}(n \log n)$。

J. Rikka 与子集 IV

题目描述

给定一棵 n 个点的树，对 $i = 1, 2, \ldots, n$ 求大小为 i 的连通块（连通导出子图）数量。

数据范围：$1 \le n \le 10^5$

题解

设多项式 $dp_u(x)$ 中 x^i 的系数表示在 u 的子树内包含 u 且大小为 i 的连通块数量，则有 $dp_u(x) = 1 + x\prod_{v \in son_u} dp_v(x)$，答案即 $ans(x) = \sum_{i=1}^{n} dp_i(x)$。直接计算可以算出所有 $dp_u(x)$ 但不能在可接受的复杂度下算出 $ans(x)$。

考虑用重链剖分优化，定义以下多项式辅助转移和计算答案：

$sum_u(x)$ 表示 u 子树内除 u 以外的点的 $dp(x)$ 之和：$sum_u(x) = \sum_{v \in subtree_u, v \neq u} dp_v(x)$。

$g_u(x)$ 表示 u 的所有轻儿子的 $dp(x)$ 之积：$g_u(x) = \prod_{v \in lightson_u} dp_v(x)$ 。

$f_u(x)$表示 u 的所有轻儿子的 $sum(x)$ 之和：$f_u(x) = \sum_{v \in lightson_u} sum_v(x)$ 。

设 son 表示 u 的重儿子，则可以把转移写作矩阵乘法形式：

$$\begin{pmatrix} dp_u(x) \\ 1 \\ sum_u(x) \end{pmatrix} = \begin{pmatrix} x \cdot g_u(x) & 1 & 0 \\ 0 & 1 & 0 \\ 1 & f_u(x) & 1 \end{pmatrix} \begin{pmatrix} dp_{son}(x) \\ 1 \\ sum_{son}(x) \end{pmatrix}$$

显然，计算 $g_u(x)$ 和 $f_u(x)$ 的时间复杂度仅和 u 的轻子树大小有关，在链顶计算一条重链的所有转移矩阵乘积时仅与重链上所有点的轻子树大小有关，故重链剖分能优化上述转移。

答案即 $dp_{root}(x) + sum_{root}(x)$。

总复杂度为 $O(n \log^3 n)$。

注意到在链顶计算一条重链的所有转移矩阵乘积时，矩阵中仅四个位置的多项式会发生变化，因此可以只维护这四个位置的多项式以优化常数。

K. 天天爱跑步

题目描述

给一棵基环树，对每个点 i 求过 i 的最长简单路径。

数据范围：$1 \le n \le 10^5$。

题解

首先对于树的情况，先 dfs 一次，求出每个点往子树走的最长路径。那么经过一个点的最长路径要么是两个子树的最长路径拼起来，要么是子树和往上走的路径拼起来，可以通过换根 DP，维护往上的最长路径，dfs 两次求出答案。

对于基环树的情况。首先找到环，对环上每个点的非环子树 dfs，求出每个点往子树走的最长路径 mxd，然后考虑对于每个环找到往其他环走的最长路径 $other_i$，那么对于 i 就是找到最大的 $mxd_j + \max(|j - i|, |n - j + i|)$，这个可以通过拆系数或者拆环倍长，双指针维护。

那么先不考虑对于环上的点，一条路径连接其他两个环上的点的子树，经过这个点的情况。对于其他所有情况，都可以视作上面树的情况，拿着 $other$ 往子树 dfs 换根即可。

那么就只需要考虑其他两个环上点经过这个环上点的情况。先拆环，相当于有一个序列。分两种情况：

- $mxd_i - i + mxd_j + j$ 对区间 $[i, j]$ 有贡献
- $mxd_i + i + mxd_j - j$ 对区间 $[1, i], [j, n]$ 有贡献

这两种情况都可以通过维护前缀后缀最大值解决。

时间复杂度：$O(n \log n)$