题解

BDFZ NOIP 2020 day2

Elegia

2020 年 11 月 27 日

目录

1　生活在树上 (tree)

我们注意到想要达到这个下界，等价于一件事：对于每个值，每次交换必然将它挪动到它目的地的方向。因此任取一组合法解，我们考虑初态的编号最小的边，满足两端点的值都想要到对面去，那么这两个点的第一次交换必然是在这条边上，且与之前的操作均不影响，故可以令其为第一次操作。我们之间用一个堆来维护当前可以操作的边就可以了。

时间复杂度 $\Theta(n^2 \log n)$。

2　三千世界 (thousands)

对于一颗树上的一些路径组成的集合 S，定义 $f(S)$ 为最大的子集 $T \subseteq S$ 的大小，满足 T 内的路径点两两不交。我们认为点对 (x, y) 表示了一条路径。

对于全体路径 $P = \{(x, y) | 1 \leq x, y \leq n\}$，对

$$\sum_{S \subseteq P} f(S)$$

求和。

2.1　暴力 $O(?)$

略。

2.2　observation

我们考虑一个路径集的 f 值是怎么确定的。可以认为是自底向上贪心，将 lca 自底向上尝试能不能放。在自下而上的过程中，我们可以认为一条路径被选择了，那么这个 lca 所在的子树就被全部剥离，剩下有用的路径必然是两端点都还未被剥离的。

2.3　正解 $O(n^2)$

我们首先转为求期望，最后乘以 2^{n^2}。考虑一个树形 DP。

我们记 $f(u, k)$ 是自下而上处理到 u 之后，未被剥离的点数为 k 的概率（显然这是和根连通的，$k = 0$ 表示 u 已经被一条选择的路径覆盖了）在树形 DP 合并的时候如果一个 k 与 k' 大小合并，那么就有 $2^{-2kk'}$ 的概率没有出现任何有效的路径，此时贡献给 $k + k'$，否则贡献给 0。注意到这是一个树上背包，所以是 $\Theta(n^2)$ 的。

3　大过滤器 (filter)

3.1　反转路径 $O(m \log m \log k)$

注意到每条边的实际权值都是可以通过所在层数直接算出来的，实际上在第 i 层内部的边就是 2^{k-i} 长度，否则就是 0 边。用一些可持久化线段树哈希之类的方法可以维护整数的加法以及高效比较。

3.2　有理有据的剪枝 $O(m \log m)$

我们考虑按层跑最短路，每层所有点的最短路算出来之后再传到下一层。注意到此时我们将本层的所有数同时减去一个值，对后续的最短路是没有影响的，不妨减去最小值，接下来本层就是从 0 开始了。我们注意到不妨设一个阈值 T，设传入的距离从小到大排列为 $d_1, \ldots, d_l$，那么若 $d_{i+1} - d_i > T$，我们可以对所有 $j > i$，将 d_j 减去 $d_j - d_i - T$。这为什么是对的呢？因为这个 $> T$ 的距离值和这个解的差距 Δ 每经过一条边要么 ± 1，要么乘以 2，如果 $T \geq m$ 的话，那么这个 Δ 是永远 ≥ 0 的了，因此这么调整之后不会改变答案。

所以我们存的数都只需要在 $n \cdot T$ 范围内了，没有了高精度。然后跑最短路即可。

3.3　优化 $O(m)$

假设本层初始的 d 值已经是排好序的，我们将其维护一个队列。将最短路引发的更新的距离值维护另一个队列。每次取两个队列中较小的 `top` 来更新，那么可知后者这个队列始终是单调的，同时我们也维护了结果 d 值的排序情况。这样我们就避免了堆的使用。

4　碎梦 (broken)

通过一个形如第二类斯特林数的递推式，其中有 k 个位置被强制赋值了，问其他部分正常递推得到的 (n, m) 位置点值。

4.1　暴力 $O(nm)$

按照题意模拟即可。

4.2　加速 $O(km^2 \log n + m^3 \log n)$

如没有修改，由线性变换可以构造转移矩阵

$$\mathbf{v}_{n+1} = \mathbf{M}\mathbf{v}_n, \mathbf{M} = \begin{bmatrix} 0 & & & & \\ 1 & 1 & & & \\ & 1 & 2 & & \\ & & \ddots & \ddots & \\ & & & 1 & m \end{bmatrix}$$

我们只需通过矩阵快速幂转到下一个进行修改的位置即可。由于我们过程中做的总是矩阵乘向量，所以通过这一点进行预处理，可以在 $O(m^2 \log n)$ 的时间内解决一个修改。

4.3　贡献计算 $O(km \log n + k^2 m)$

我们不妨将赋值操作看做加法，也就是在 (x, y) 位置加上 $z - f(x, y)$，其中的 f 是赋值之前的值。这样的话计算 (x_i, y_i) 的时候，我们只需考虑每个之前的 (x_j, y_j) 位置对其的贡献。

我们不妨考虑递推式的组合意义，(x, y) 有 y 种方案走到 $(x + 1, y)$，有 1 种方案走到 $(x + 1, y + 1)$，从 (x_j, y_j) 走到 (x_i, y_i) 的方案数就是假设 $1 \sim x_j$ 已经

被划分成了 y_j 个集合，那么接下来的 $x_j+1\sim x_i$ 还有多少种划入集合的方式，使得最后总共有 y_i 个集合。那么我们可以认为是有 x_i-x_j 个数，他们的取值在 $1\sim y_i$，且 $y_j+1\sim y_i$ 必须都有。为了去掉顺序关系，最后还要乘以 $\frac{1}{(y_i-y_j)!}$。因此 (x_j,y_j) 对 (x_i,y_i) 的贡献可以用容斥得到

$$\frac{1}{(y_i-y_j)!}\sum_l(-1)^l\binom{y_i-y_j}{l}(y_i-l)^{x_i-x_j}$$

你可以通过 $O(km\log n)$ 甚至 $O(km\log_m n)$ 的时间预处理快速幂（通过线性筛，不过数据范围并没有弄这么紧）。然后每个 (x,y) 对之间的贡献可以 $O(m)$ 计算。因此复杂度为 $O(km\log n+k^2m)$。