A

设 f_u 为 u 子树的答案，T_u 为 u 子树内的所有边权之和，W_u 为 u 子树内所有点权之和，则假如已经确定了遍历儿子的顺序 $v_1, v_2, \ldots, v_k$。

那么转移式为 $f_u = \sum_{i=1}^{k} \left(f_{v_i} + W_{v_i} \times \sum_{j=1}^{i-1} T_{v_j} \right)$。

考虑交换相邻两个儿子 v_i, v_{i+1} 发生的变化，显然 v_i, v_{i+1} 之前以及 v_i, v_{i+1} 之后的贡献部分不会变化，变化的是 v_i, v_{i+1} 带来的贡献：

1. 当遍历顺序 v_i 先于 v_{i+1} 时，这部分贡献为 $f_{v_i} + W_{v_i} \times \sum_{j=1}^{i-1} T_{v_j} + f_{v_{i+1}} + W_{v_{i+1}} \times \sum_{j=1}^{i} T_{v_j}$
2. 当遍历顺序 v_{i+1} 先于 v_i 时，这部分贡献为
   $f_{v_{i+1}} + W_{v_{i+1}} \times \sum_{j=1}^{i-1} T_{v_j} + f_{v_i} + W_{v_i} \times \left(\left(\sum_{j=1}^{i-1} T_{v_j} \right) + T_{v_{i+1}} \right)$

则遍历顺序 $v_i v_{i+1}$ 先于 $v_{i+1} v_i$ 时，需要满足的条件为

$$f_{v_i} + W_{v_i} \times \sum_{j=1}^{i-1} T_{v_j} + f_{v_{i+1}} + W_{v_{i+1}} \times \sum_{j=1}^{i} T_{v_j} < f_{v_{i+1}} + W_{v_{i+1}} \times \sum_{j=1}^{i-1} T_{v_j} + f_{v_i} + W_{v_i} \times \left(\left(\sum_{j=1}^{i-1} T_{v_j} \right) + T_{v_{i+1}} \right)$$

$$f_{v_i} + f_{v_{i+1}} + W_{v_i} \times \sum_{j=1}^{i-1} T_{v_j} + W_{v_{i+1}} \times \left(\left(\sum_{j=1}^{i+1} T_{v_j} \right) + T_{v_i} \right) < f_{v_i} + f_{v_{i+1}} + W_{v_i} \times \left(\left(\sum_{j=1}^{i+1} T_{v_j} \right) + T_{v_i+1} \right) + W_{v_{i+1}} \times \sum_{j=1}^{i-1} T_{v_j}$$

$$W_{v_{i+1}} T_{v_i} < W_{v_i} T_{v_{i+1}}$$

然后就将 sort 里面的 cmp 函数填上这个就可以了。

为什么将儿子序列排一遍序是正确的？我们考虑任一儿子序列 $v_1, v_2, \ldots, v_k$，根据冒泡排序的原理，每次选出两个相邻的逆序对并交换，可以使逆序对刚好 -1，直至没有相邻的逆序对，说明序列已排序，而在本题中将两个相邻的不优对进行交换，每次交换都会使答案变得更优。直至没有相邻的不优对，说明儿子序列也已按照某种特定的顺序进行了排序，而对于任意一种序列，都可以做若干次使答案更优的交换操作使其变成这种特定顺序的序列，说明这种序列的顺序就使所有排列方式中最优的，故直接排序是完全正确的。

时间复杂度：$O(n \log n)$。

B

考虑倒着枚举操作序列，其中 $[r_i + 1, r_i + 1 + (r_i - l_i)]$ 是由 $[l_i, r_i]$ 复制过来的，对于这部分字符我们只需要按顺序枚举它是由哪个字符复制过来的即可，用 fa_i 来存储字符 i 是由原先的哪个字符复制过来的。

显然只需要考虑最终位置编号 $\leq r$ 的字符即可，设 now 为当前最终位置编号 $\leq r$ 的字符还剩下多少，每次解决掉一个字符是从哪里复制的就将 now 减一，当 $now \leq r_i$ 时说明当前处理的字符的最终编号 $> r$，并不在我们输出的范围之内，无需再探究它是由谁复制而来的了。

快速维护每个字符所代表的编号可以用树状数组维护，查询一个当前编号所对应的最终编号可以用树状数组上二分解决。

然后根据 fa_i 和原字符串 s 推出新的答案字符串即可。

时间复杂度：$O(n \log n + q)$，常数很小。

C

$V \leq 21$

注意到两个相同的数相乘可以互相抵消掉，所以每个数可分类成**选奇数个**和**选偶数个**两种状态，选奇数个可以看成选一个，选偶数个可以看成不选，那么最大的乘积只有 $V! = 51090942171709440000$，可用 `__int128` 存下，判断是否为完全平方数只需要试除 $1 \sim V$ 内的所有质数平方因子即可。

时间复杂度：$O(2^V \times V)$。

$V \leq 33$

注意到 $\mu(i) = 0$（即含平方因子）的数可以通过不断除以平方因子做到与 $\mu(i) \neq 0$ 的数本质相同，所以无需考虑 $\mu(i) = 0$ 的数。而 V 以内的 $\mu(i) \neq 0$ 个数最多只有 21 个，最大的乘积也只有 $3005152014522601944000 0$，也可以用 `__int128` 存下。

时间复杂度：$O\left(2^{\sum\limits_{i=1}^{V} |\mu(i)|} \times \sum\limits_{i=1}^{V} |\mu(i)|\right)$

$V \leq 69$

注意到 $\mu(i) \neq 0$ 可以表示成若干个质数（且次数均为 1）相乘，而 $\omega(V) \leq 19$，所以每个数的质因数分解状态可以用 19 位的二进制数来表示，完全平方数要求选择的若干数所代表的状态的**异或和** $= 0$，直接跑 状压dp 即可。设 $f_{i,S}$ 代表 $1 \sim i$ 内的所有数中选一些数使得这些数所代表的状态的异或和为 S 的方案数，转移是平凡的。

时间复杂度：$O\left(2^{\omega(V)} \sum\limits_{i=1}^{V} |\mu(i)|\right)$

$V \leq 139$

注意到选奇数个 $> \dfrac{V}{2}$ 的质数**没有任何意义**，所以这部分直接让答案乘上选偶数个 $> \dfrac{V}{2}$ 的质数的方案数即可。

时间复杂度：$O\left(2^{\omega(\frac{V}{2})} \sum\limits_{i=1}^{\frac{V}{2}} |\mu(i)|\right)$

$V \leq 312$

考虑设阈值 B 然后折半，当一个数不存在 $> B$ 的质数的时候跑状压 dp，存在 $> B$ 的质数的数很少，可以暴力搜索这些数每个数选奇数个还是选偶数个。

当 B 取 69 时，一共有 43 个数存在 > 69 的质数，其中质数在 $[71, 103]$ 之间的有 $p, 2p, 3p$，这部分的质数个数有 8 个，质数在 $(103, 156]$ 之间的有 $p, 2p$，这部分的质数个数有 9 个，所以有效的搜索量只有 $4^8 \times 2^9 = 2^{25} = 33554432$，可以通过。

$V \leq 4021$

考虑根号分治，由于将一个数质因数分解后只存在一个 $> \sqrt{V}$ 的质数，所以可以根据这个质数是什么（或不存在）分类，每一类必须选择偶数个数，然后将 $< \sqrt{V}$ 的质数状压做 dp 即可，由于每一类要选择恰好偶数个数，所以要在原来的 dp 基础上多开一维 $0/1$ 代表该类数选了奇数个还是偶数个。

由于 $\omega(\sqrt{V}) \leq 18$，且第 18 个质数 61 因为有 $61 \times 67 = 4087 > V$，所以可以把 61 也看成大质数，所以质数只有 17 个，$2^{17} = 131072, \sum\limits_{i=1}^{V} |\mu(i)| \leq 2446$，二者相乘为 $320602112 \approx 3.2 \times 10^8$。

$V \le 1234$ 给想出根号分治但是实现的复杂度较劣的算法，例如对每一维单独做背包然后合并，使用了快速幂求转移时候的 2 的幂次等等。

时间复杂度：$O\left(2^{\omega(\sqrt{V})}\sum\limits_{i=1}^{V}|\mu(i)|\right)$

$V \le 5123$

按照 $V \le 4021$ 的思路，小质数有 19 个，$2^{19}=524288,\sum\limits_{i=1}^{V}|\mu(i)|\le 3115$，二者相乘为 $1633157120\approx 1.63\times 10^9$ 很难过去。

但是我们发现，大质数很大的时候，小质数就只能很小，所以我们可以先枚举较大的大质数类，这样做状压 dp 的时候 S 只需要开到很小的范围即可，当枚举的大质数不断变小的时候，S 再不断变大直至 $[0,2^{19})$ 范围，经测试，该做法的耗时不到上面做法的 $\dfrac{1}{5}$，可以通过。

由于选奇数/偶数个数的方案数均需要用的 2 的幂次，而由于 $\mu(i)=0$ 的 b_i 会加到 $\mu(i)\neq 0$ 的 b_i 中，所以最大的 b_i 如 b_1 可以达到 $10^5\times\sqrt{V}\le 7.1\times 10^6$，所以你需要预处理 7.1×10^6 以内的 2 的次幂。

Bonus：$V\le 10^6$

drdilyor Orz 把标算给踩了。

我们发现当 $b_i>0$ 时选奇数个和选偶数个方案数一样，这其实就是个固定的系数。

而选择若干个数使得异或和为 0 的方案数，这其实就是 2 的把它们插进线性基里的自由元个数次幂。

再套用根号分治以及上述优化外加线性基本就支持的 bitset 优化，可以做到：$O\left(\dfrac{V^2}{\omega\ln^2 V}\right)$，毛估估一下可以过 10^6。

D

$n\le 100,m=1,k\le 8$

相当于只有第一行的数互不相同，以及第一列的两个数要求不相同。

如果只考虑第一条限制，那么因为要求所有格子异或和为 0，无论第一行怎么选第二行都有且仅有一种情况使得所有格子异或和为 0，所以这部分的方案数为 $(2^k)^{\underline{n}}$。

但是这样第一列的两个格子的值可能会相同，这也就意味着剩下的 $n-1$ 个格子的值互不相同且异或和为 0，可以按值域顺序 dp，设 $f_{i,j,S}$ 为 $1\sim i$ 中选了 j 个数异或和为 S 的方案数，转移是平凡的。

时间复杂度：$O(Tn\times 4^k)$

$m=1$

设 f_i 为 i 个不同 k 位二进制数（有序）使得异或和为 0 的方案数，那么先随便钦定前 $i-1$ 个 k 位二进制数这样第 i 个二进制数也就确定了，总方案数为 $(2^k)^{\underline{i-1}}$。但是可能会出现第 i 个二进制数和其中某个二进制数相同的情况，这种需要减去，除去这两个数还剩下 $i-2$ 个数，这 $i-2$ 个数是互不相同的且异或和为 0，方案数为 f_{i-2}，由于第 i 个数不能和其它 $i-2$ 个数相同，所以可选的方案数共有 2^k-i+2 种，同时与第 i 个数相同的位置还不确定，所以还需要乘上 $i-1$。由此可以得出 f_i 的递推式：

$$f_0=f_1=1$$
$$f_i=(2^k)^{\underline{i-1}}-f_{i-2}\times(2^k-i+2)\times(i-1)\ (i\ge 2)$$

若第二行的某个格子和第一行的某个格子的值相同，则称这两个格子配对，显然这两个格子不能处在同一列。

关于 $m=1$ 还有一种做法，对第二行的第一个格子是否与第一行的某个格子配对分类讨论。

1. 若第二行的第一个格子不与第一行的任何格子配对，则这 $n+1$ 个数均不相同，方案数为 f_{n+1}。
2. 若第二行的第一个格子与第一行的某个格子配对，则剩下的 $n-1$ 个格子均不相同且异或和为 0，这部分方案数为 f_{n-1}，第二行的第一个格子跟任意一个格子（除第一行的第一个格子）都可以相同，方案数为 $n-1$，这一对格子必须和剩下的 $n-1$ 个格子不相同，方案数为 2^k-n+1，所以总方案数为 $f_{n-1}\times(n-1)\times(2^k-n+1)$。

最后将二者方案数加起来即可。

时间复杂度：$O(Tn)$。

$m\le 7$

还是枚举第二行的格子是否与第一行的格子配对以及跟第一行的哪个格子配对，但因为配对要求位置错排。所以第二行的格子与第一行的前 m 个格子配对可能会影响之后的格子配对的选择空间，故可以将第二行每个格子的配对种类分为 $m+2$ 种：不配对，与前 m 个格子的其中一个配对，与前 m 个格子之后的格子（本质相同）配对，设成功配对了 x 对，其中有 y 对是跟第一行前 m 个格子之后的格子配对的，则方案数为 $f_{n+m-2x}\times(2^k-(n+m-2x))^{\underline{x}}\times(n-m)^{\underline{y}}$。

预处理 2^k 的下降幂及其逆元（这部分需要线性），阶乘及其逆元即可 $O(1)$ 算出方案数。

设 $F(m)$ 为本质不同的配对方案数（其中 $F(7)=362384$），则时间复杂度为：$O(T(mF(m)+n))$

$m=n+1$

此做法跟正解没有关联。

这说明一个很重要的性质：$n+m$ 是奇数！

思考一下，假设当前纸条的所有格子的值的异或和为 x，那么给所有格子均异或上 x，总异或和也会异或上 x 从而变成 0，相同行，列之间的互不相同条件该满足还是会满足。

显然对于每种总异或和，满足相同行，列之间的互不相同条件的方案数一定相等，所以我们只需要统计出满足互不相同条件的方案数最后除以 2^k 就是答案。

我们先可以填好第二行，方案数为 $(2^k)^{\underline{m}}$，第一行考虑容斥，设 $f(S)$ 为只有集合 S 内的每一列元素相同的方案数，$g(T)$ 为钦定集合 T 内的每一列元素相同的方案数，则有：

$$f(S)=\sum_{S\subseteq T\subseteq\{1,2,\dots,n\}}(-1)^{|T|-|S|}g(T)$$

显然对于相同的 $|T|$，$g(T)$ 必然相同，则满足条件的第一行的方案数为：

$$f(\emptyset)=\sum_{T\subseteq\{1,2,\dots,n\}}(-1)^{|T|}g(T)=\sum_{i=0}^{n}(-1)^i\binom{n}{i}(2^k-i)^{\underline{n-i}}$$

时间复杂度：$O(T(n+m))$

正解

在 $m\le 7$ 的做法中我们发现其实只需要关心**成功配对了多少对**以及**每类被认为本质相同的配对方案一共有多少种**。

在这里我们认为**两个配对方案本质相同**当且仅当**配对的对数相同**，所以不妨枚举成功配对了 i 对，本质相同的配对方案数为 $d_i \times \binom{m}{i}$ 种，那么方案数则为 $f_{n+m-2i} \times (2^k - (n+m-2i))^{\underline{i}} \times \binom{m}{i} \times d_i$ 。

设 g_i 为有多少 n 阶排列 $[p_1, p_2, \ldots, p_n]$ 满足对于 $1 \le j \le i$ 都有 $p_j \neq j$ 的方案数（类似错排状物），则 $d_i = \dfrac{g_i}{(n-i)!}$。

这里不妨认为 $n \ge m$，则 $g_n = D(n)$，考虑倒着递推，若现在需要从 g_i 递推到 g_{i-1}，有两种情况：

1. 若 $p_i \neq i$ 则方案数刚好为 g_i。
2. 若 $p_i = i$ 则可以直接把 p_i 删除，方案数为有多少 $n-1$ 阶排列 $[p_1, p_2, \ldots, p_{n-1}]$ 满足对于 $1 \le j \le i-1$ 都有 $p_j \neq j$ 的方案数，设其为 h_i。

由此可以得出 $g_{i-1} = g_i + h_i$。

现在考虑 h_i 如何递推，关于普通错排有一个经典公式： $D(n) = (n-1)(D(n-1) + D(n-2))$，做法是 p_{p_n} 是否 $= n$ 分类讨论：

1. 若 $p_{p_n} = n$，则问题变成了大小为 $n-2$ 的错排问题，方案数为 $D(n-2)$。
2. 若 $p_{p_n} \neq n$，则问题变成长度为 $n-1$ 的排列中，对于任意的 $1 \le i \le n-1$ 都需要满足 $p_i \neq i$，这是大小为 $n-1$ 的错排问题，方案数为 $D(n-1)$。

因为 p_n 的取值有 $n-1$ 种，所以 $D(n) = (n-1)(D(n-1) + D(n-2))$。

那么拓展一下，设 $D(n, m)$ 为长度为 n 的排列，前 m 个元素要求错排的方案数。

那么做法还是一样，设 $p_x = m$ $(x \neq m)$，则删掉 p_m 之后需要让 p_x 继承原来的 p_m，接下来对 x 的大小分类讨论：

1. 若 $x < m$（这样的 x 共有 $m-1$ 种），则可能会导致 $p_x = x$，但这并没有关系，我们可以不强制 $p_x \neq x$，这样问题变成了长度为 $n-1$ 的排列，前 $m-2$ 个元素要求错排的方案数，即 $D(n-1, m-2)$。
2. 若 $x > m$（这样的 x 共有 $n-m$ 种），则无论 p_x 是否 $= x$ 其实没影响，因为此前我们并没有强制 $p_x \neq x$，这样问题变成了长度为 $n-1$ 的排列，前 $m-1$ 个元素要求错排的方案数，即 $D(n-1, m-1)$。

由此我们可以得出递推式 $D(n, m) = (n-m) \times D(n-1, m-1) + (m-1) \times D(n-1, m-2)$，转化成 g_i, h_i 的关系式就是：

$$g_i = (n-i) \times h_i + (i-1) \times h_{i-1}$$

由此可以得出根据 g_i, h_i 递推出 h_{i-1} 的关系式：$h_i = \dfrac{g_i - (n-i)h_i}{i-1}$。

综上，可以得出如下的 g_i, h_i 递推式：

$$g_{i-1} = g_i + h_i$$
$$h_{i-1} = \frac{g_i - (n-i)h_i}{i-1}$$

可以在 $O(n)$ 时间内得出所有 g_i, h_i，进而得到所有 d_i。

时间复杂度：$O(T(n+m))$

时限开的比较松，只要写的是严格线性做法应该都能过，如果用 $O(\log \mathrm{mod})$ 在线处理逆元只会被卡成 88 分。