最小生成链 Solution

首先简单地说一下前两档部分分的做法：

- ***Subtask* #1**：$n \leq 8$，枚举每条链，得到最小值即可。时间复杂度 $\mathcal{O}(n!)$。
- ***Subtask* #2**：$n \leq 17$，类似于旅行商问题的做法，状压 DP 维护一个数组 $f[u][S]$ 表示目前这条链延伸到节点 u，链中包含的节点集合为 S 时，链中最大值的最小值。每次 $\mathcal{O}(n)$ 枚举下一个节点直接转移过去即可。时间复杂度 $\mathcal{O}(n^2 2^n)$，由于遍历到的状态不多，是可以通过 $n = 17$ 的数据的。

剩下的两档部分分做法我将在下面介绍正解的过程中提到。

原问题在完全图上，边数达到了 n^2 级别。直接在原图上处理显然是不行的，因此考虑转化问题。

众所周知，我们可以从左往右唯一的遍历一条链，这样得到的遍历序实际上就是一个序列。因此，一条链与一个序列是可以一一对应的。也就是说，对于一个 n 个点若干条边的图，它当中存在的所有生成链对应的就是 1 到 n 的一个排列，且这个排列满足相邻元素间在原图中有边。

由于题目中给出的图是完全图，因此 1 到 n 的所有 $n!$ 种排列都是符合要求的。于是原题就可以转化为：

> 给出一个长度为 n 的序列 $a_1, a_2, \cdots, a_n$，请找出一个排列，使得相邻两元素间异或值的最大值最小。

现在我们看到 ***Subtask* #4** 的特殊性质：序列中的元素只有 0 或 1。

相信大家都能推得出来这个子任务的结论：当序列中所有元素都是 0 或都是 1 时，答案为 0；否则——序列中既有元素 0 又有元素 1 时，答案为 1。

为什么会这样呢？感性理解一下，就是原序列中既有 1 也有 0，那么无论如何都避免不了有至少一组 0 和 1 相邻，因此最大值至少就为 1。

我们把这个情况推广一下。由于位运算任意两位是不会互相影响的，我们考虑提取出原数列中的第 i 位，单独考虑这一位的情况。我们容易发现，对于这独立的一位，同样满足类似性质——当原序列中所有数在这一位上都是 0 或都是 1 时，这位的答案一定是 0；否则这位答案一定是 1。

相信大家可以已经有一些想法了。

接下来我们需要回想起进制的一个有趣的性质，这也是我们经常利用的来做贪心的性质。也就是对于该进制的某一位 i，如果有两个数 a, b，满足 a, b 高于 i 的位全部相等并且 a 在第 i 位比 b 在第 i 位上的数大，那么无论低于 i 的位是什么样的，a 一定大于 b。

二进制也满足这个性质。因此我们考虑找出最高的一位满足原数列中所有数在这一位上既有 0 也有 1，那么我们就可以根据这一位的值将原序列分成这一位为 0 和这一位为 1 的两部分。我们不难发现，对于这每个部分内，由于这个最高位是 0，因此永远不可能出现最大值。这个序列的最大值一定是出现在 0 和 1 交界的部分。

我们希望这个交界的部分两元素异或值最小。也就是说，我们需要从这一位为 0 的元素中和这一位为 1 的元素中各找出一个元素，使得这两个元素的异或值最小。

于是有一个显而易见的 $\mathcal{O}(n^2)$ 做法，枚举任意两个元素，找出异或值的最小值。解决了 ***Subtask* #3**：$n \leq 1\,000$。

接下来这个找最小值的过程可以进行优化。这里有异或操作，很自然地能够想到 0-1 Trie。我们可以维护一棵 0-1 Trie，我们将序列中所有这一位为 0 的元素插入这棵 Trie，然后用所有这一位为 1 的元素去 Trie 中查异或最小值。最后所有最小值的最小值就是答案了。

要注意特判一下所有元素都相同的情况，因为这会找不到这个最高的位满足这一位上有 0 和 1。

时间复杂度和空间复杂度都是 $\mathcal{O}(n \log a_i)$。

穿越 Solution

首先简单地说一下某些部分分的做法：

- $n, m \leq 300$：$\mathcal{O}(n^2 m)$ 的 DP，用 `f[i][j][k]` 表示经过 i 步，从 j 号点出发，达到 k 号点的方案数。
- $n \leq 200, m \leq 10^6$：将上面的 DP 转移式子写出来，发现很类似一个矩阵乘法的形式。所以可以直接维护原图的邻接矩阵，求出这个矩阵的 m 次幂，第 i 行第 i 列的数就是第 i 个点的答案。时间复杂度 $\mathcal{O}(n^3 \log m)$。
- 所有 x_i 相等：观察原图，发现只有 x_i 与 0 之间有一个二元环。而且 x_i 和 0 都有自环。考虑对于每一步，都有可能走到另外一个节点或走回自己，而最后一步必须走回自己，因此答案为 x_i 和 0 的答案 2^{n-1}，其它点答案都是 0。时间复杂度 $\mathcal{O}(n)$。
- 对于所有的 $1 \leq i \leq n$，都有 $x_i = i$：与所有 x_i 相等的部分分类似，唯一不同之处在于其它点答案都是 1。

对于 $n, m \leq 2\,000$，出题人 X 某人和出题人 T 某人各自提出了一个 $\mathcal{O}(nm)$ 的做法。

X 某人的 $\mathcal{O}(nm)$ 做法

注意到每次转移必定有一个点 0，假设我们知道 0 号点到 i 号点走 $m-j$ 步的路径条数，我们就可以对这条出边直接计算答案。

预处理出 0 号点到每个点走 j 步的路径条数，对于每个点，从该点开始，直接得出走通向 0 的出边的贡献，暴力跑另一条出边，复杂度 $\mathcal{O}(nm)$。

T 某人的 $\mathcal{O}(nm)$ 做法

直觉告诉我们，0 号点是一个非常特殊的点，每个点都有一条边可以直接通向 0 号点。因此我们考虑将最终的路径分为两种不同的路径：经过 0 的路径和不经过 0 的路径。

如果一条路径不经过 0，那么这条路径一定不会经过连向 0 的那条边。那么我们将所有连向 0 的边去掉，只留下所有不连向 0 的边。发现这张剩下的图一定是一个基环内向森林，也就是说图中的每个联通块都是一个基环内向树。由于保证了所有 x_i 都非零（这是一个很重要的条件），因此 0 一定不在基环内。

那么对于这个基环内向树，对于一个节点 u，有一条不经过 0 的路径满足走 m 步回到 u，当且仅当 u 在一个基环内，且 m 是环长 len 的倍数。而且这种路径要么不存在，要么只有一条。

如上可以特判一下，然后用拓扑排序之类的东西找个环，再用个 DFS 求环长，这里的时间复杂度是 $\mathcal{O}(n)$ 的。

否则路径就一定经过点 0。我们考虑将这条路径分成两个部分分别统计方案数，最后通过乘法原理乘起来：

- 从 u 开始，第一次到达 0 的这一段路径；
- 从第一次到达 0 开始，走若干步（可以经过 0），最终回到 u 的这一段路径。

对第二段路径，直接在原图上从 0 开始跑一遍 DP 就可以了。准确来说，用 `f[i][j]` 表示从 0 号点出发，经过 i 步，达到点 j 的方案数。这里复杂度是 $\mathcal{O}(nm)$ 的。

对第一段路径，将原图所有边反向，在反图上跑一遍类似的 DP 即可。仍然是 $\mathcal{O}(nm)$ 的。

这样我们就可以通过枚举断点——第一次到达 0 的步数——来合并答案了。合并可以做到 $\mathcal{O}(m)$。

最终时间复杂度也是 $\mathcal{O}(nm)$。

接下来介绍本题的线性做法。

同样将路径分类，分为经过 0 的路径和不经过 0 的路径。不经过 0 的路径可以快速统计，复杂度瓶颈在于经过 0 的路径。因此我们只考虑对经过 0 的路径条数进行优化。

首先我们发现，一个点 u，有可能有一条路径经过 0 回到自己，当且仅当点 u 与 0 在同一棵基环内向树内。

我们将一个点 u 回到的自己的，经过 0 的路径条数分成四段统计：

- 从 u 开始，第一次到达 0 的这一段路径；
- 从第一次到达 0 开始，走若干步，最终最后一次回到 0 的这一段路径；
- 从最后一次回到 0 开始，沿着基环内向树上的最短路一直走，回到点 u 的这一段简单路径；
- 从回到点 u 开始，只走点 u 所在的环，最终回到点 u 的路径。

第四段存在，当且仅当 u 在这个环上。如果 u 不在环上，则不存在第四段路径，需要特判。

我们先看到第二段路径。我们仍然把所有连向 0 的边去掉，剩下一个基环内向森林，并且我们只考虑 0 所在的这棵基环内向树。注意到从 0 出发，可以无限走下去（因为可以在环上走任意步），而在每一步都可以走这条连向 0 的边回来，在自身也可以直接走这条连向 0 的边回来。设我们在第二段路径上走了 k 步，我们发现这很类似于一个背包的过程：可以选取 1 到 $+\infty$ 中任意重量的物品，考虑物品的先后顺序，求最终重量恰好为 k 的方案数。

这个方案数是非常好求的。实质上就是有 k 个小球，在这些小球中的 $k-1$ 个空隙中插入任意多个板，求方案数。答案其实就是枚举每个位置上是否插板：2^{k-1}，特殊地，当 $k=0$ 时，答案也是 1。

然后再考虑第三段路径。这段路径应该是唯一的，长度可以预处理出来。这段路径的用处只是算出剩下有多少步可以自由地分配。

然后看到第一段路径。跟第二段路径类似，也是可以无限走下去，而且在每一步也都可以通过一条连向 0 的边直接到达目的地，所以这里对方案数的贡献实际上是第二段路径的出来的方案数的前缀和。形式化地说，我们不妨设 $f(u,k)$ 表示除去第四段路径以后，需要走 k 步，从 u 号点出发回到自身的方案数。那么我们有如下式子成立：$f(u,k)=2^{k-len(0,u)}$，其中 $len(0,u)$ 表示在基环树上从 0 出发，到 u 的最短路径长度。

式子的推导大概是，在第一段路径中，可以花掉任意步数，然后答案就是第二段路径的一个前缀和。我们发现第二段路径的方案数前缀和恰好也是一个 2 的整数幂。然后就得到了上述式子。

接下来考虑这个第四段路径。发现第四段路径的长度一定是点 u 所在的环的长度，并且对于所有需要求的点 u，这个环都是一样的，也就是说环的长度是一样的。那么所有的前三条路径剩下的步数模环长 $cycleLen$ 应该是同余的。因此考虑先求出所有 2 的整次幂，定义 $b_i=2^i\ (0\le i\le m)$，然后在 b_i 上分除以环长 $cycleLen$ 的得到的余数做一个前缀和，即前缀和数组 p_i 满足如下递推式：

- 当 $i<cycleLen$ 时，$p_i=b_i$；
- 当 $i\ge cycleLen$ 时，$p_i=p_{i-cycleLen}+b_i$。

那么这时候答案就应该是这个前缀和数组了。

然后随便 DFS 处理一下这个 $len(0,u)$，然后查前缀和数组的第 $m-len(0,u)$ 项，经过 0 号点的答案就求出来了。

时间复杂度 $\mathcal{O}(n+m)$。

树统计 Solution

首先简单地说一下前几个部分分的做法：

- ***Subtask* #1**：$n, m \leq 100$，我不知道有什么特殊做法，可能实现得不优秀的暴力需要三方。
- ***Subtask* #3**：$m = 0$，考虑一种贪心方法，根据到叶子节点的距离枚举每一个点 u（可以通过 BFS 实现，一开始把所有叶子节点放入队列）。若点 u 点权小于 0，则将点 u 点权增加到 0，同时父亲节点减去相应权值。如此一直做到根节点。若根节点此时点权为负，则答案为 `No`；否则答案为 `Yes`。时间复杂度为 $\mathcal{O}(n)$。
- ***Subtask* #2**：$n, m \leq 2\,000$，有可能存在一些 $\mathcal{O}(n^2 + m \log n)$ 之类的神仙做法。这里其实只要每次修改后重构一下树，然后用 ***Subtask* #3** 的暴力就能做到 $\mathcal{O}(nm)$ 的复杂度。

接下来详细介绍一下第四个和第五个子任务的做法。

***Subtask* #4**：对于第 i 条边，$u_i = 1, v_i = i + 1$。

我们设 $q = a_1 + \sum_{i=2}^{n} \min(a_i, 0)$，不难发现 q 的意义为满足其他点的值都大等于 0 的情况下，根节点最大可能的点权。那么答案为 `Yes` 的充要条件就是 $q \geq 0$。

考虑每一次修改时对 q 的变化。若修改的节点是根节点 1，那么有影响的就是 a_1 这一项，直接将 q 加上 w 即可。

否则修改的是一个叶子节点。我们考虑先把 x 号点对 q 的贡献去掉，修改完 x 号点的点权后，再将 a_x 对 q 的贡献统计进去。首先我们讨论一下这个节点修改前的值 a_x。若 $a_x \geq 0$，则对 q 值没有影响；否则这个点对 q 值有 a_x 的贡献，就需要把 q 减去 a_x。然后将 a_x 加上 w，再将 q 加上 $\min(a_x, 0)$。

时间复杂度 $\mathcal{O}(n)$。

***Subtask* #5**：m 次变化中，w_i 为正数的次数不超过 400 次。

当 w_i 为正数时，因为只有 400 次，不妨暴力重构树；当 w_i 为 0 时，对答案没有影响。因此我们只要考虑 w_i 为负数的情况如何快速处理就行了。

回想一下先前 ***Subtask* #3** 的做法，从叶子节点开始往上不断从父亲借点权。注意到我们借点权的过程中，每进行一次操作，就会多一个点的点权变成 0。也就是说，算法的时间复杂度是和树上 0 的个数成正比的。

考虑维护一个并查集，表示该点祖先上权值不为 0 的最近的点，然后每次施魔法的时候可以施展到这个点（显然魔法是可以通过传递实现祖孙之间的交换的）。这样每条边就最多被合并一次，就只会造成一点的时间消耗。因此总时间复杂度是 $\mathcal{O}(400n + (n + m) \cdot \log n)$。

最后介绍一下正解——动态 DP。

仿照 ***Subtask* #4** 的思路，我们维护一个 DP 数组 f，f_i 表示 i 号点所在子树的值都大于等于 0 的情况下，i 号节点最大可能的点权。易证答案为 `Yes` 的充要条件就是 $f_1 \geq 0$。

然后对于 i 的每个子节点 j，若 $f_j \geq 0$，则 j 节点不需要 i 节点给它点权，则对 f_i 没有影响；若 $f_j < 0$，则 j 节点需要 i 节点给它 $-f_j$ 的点权，因此会让 f_i 加上 f_j。

那么状态转移方程就是：

$$f_i = a_i + \left(\sum_{j \in son(i)} \min(f_j, 0) \right)$$

树上问题 + 动态修改不难想到动态 DP。

依照动态 DP 的套路，开一个数组 $g_i = \left(\sum_j \min(f_j, 0)\right) + a_i$，其中 j 是 i 的轻儿子（对于叶子节点 $g_i = a_i$）。然后该方程可以转化为 $f_i = g_i + f_{wson_i}$。

然后对应到矩阵上，拿纸笔推一下不难发现需要对于每个节点维护一个矩阵 $\begin{bmatrix} f_j & 0 \end{bmatrix}$，然后转移矩阵大概这样：

$$\begin{bmatrix} g_i & +\infty \\ g_i & 0 \end{bmatrix}$$

然后抄一下动态 DP 模板，树剖套线段树维护一下矩阵转移就好了。

时间复杂度 $\mathcal{O}(n + m \log^2 n)$。如果会什么全局平衡二叉树之类的算法可以优化到 $\mathcal{O}(n + m \log n)$。