mexor

设分成的集合的 mex 从大到小分别是 $a_1 \ge a_2 \ge \cdots \ge a_k$ ，集合分别是 $S_1, \cdots, S_k$ 。

不难发现在最优方案中必然有 $\forall i < j, a_i \& a_j = 0$ ，否则可以通过把 S_j 中的一些元素移动到 S_i 来改变这一点，并通过多次移动使得答案变为 $a_1 | a_2 | \cdots | a_k$ 。

其次，可以发现 a_1 一定取的越大越好，因为增大 a_1 的同时可以不改变 $a_2, \cdots, a_k$ ，而在 a_1 增大之后通过上面的方式调整 $a_2, \cdots, a_k$ 也可以使得答案不会变小。

因此可以贪心地得到最大的 a_1 ，把 a_1 已经有的 bit 标记一下，然后贪心地得到最大的 a_2 使得 $a_2 \& a_1 = 0$ ，再把 a_2 已经有的 bit 标记一下，然后再贪心得到最大的 a_3 ......显然这个过程只需要进行 $\log a$ 轮。

wbtree

因为只能在子树里选择匹配点，因此从深到浅贪心给每个白点选匹配是正确的：如果一个白点 x 不在子树里匹配当前可用的最近的黑点 y ，而是选择较深的黑点 z ，那么换成 x 匹配 y 而原来匹配 y 的白点去匹配 z ，不会使得代价增加。

反过来，可以从浅到深枚举所有黑点，只要祖先还有未匹配的白点就匹配一个最深的白点，其正确性证明也很类似。

找到最近的未被匹配的最深的白点，这可以使用并查集维护。

andgraph

把 $i \to j$ 的边拆成 $i \to b \to j$ ，其中 a_i 和 a_j 的第 b 位均为 1 。那么我们就获得了 $\log n$ 个关键点，任意两个点之间的路径都需要经过至少一个关键点。

从每个关键点出发跑一个最短路，那么询问 i, j 的最短路时就只需要枚举关键点 b ，用 $dis(b, i) + dis(b, j)$ 更新答案即可。

非常可惜，跑最短路的时间是 3 个 log ，根本过不去。

进一步优化，把算法分为两步：

- 先求出所有 b 之间的最短路。对于 b_1, b_2 ，只需要快速找到覆盖 $2^{b_1} + 2^{b2}$ 的最小的数即可。这只需要做一个 FWT 状物（高维后缀最小值），就可以 $O(n \log n)$ 得出来。
- 然后求出每个 b 到每个 i 的最短路。除掉一开始 a_i 那条边，其实就是要求 $f(a_i) = \min_{b' \in a_i} dis(b', b)$ 。这可以直接由 $f(a_i \oplus \operatorname{lowbit}(a_i))$ 递推过来，因此复杂度也是 $O(n \log n)$ 。

整体复杂度 $O(n \log n)$ 。

splitham

对于每一次分裂，都设 $dp_{i,j}$ 表示从第一部分的 i 出发，到第二部分的 j 结束，最短需要多远。

设 i 到 j 的最短路在第一部分最后一个点是 a ，而在第二部分第一个点是 b ，那么就有

$$dp_{i,j} = \min_{a,b} dp_{i,a} + dp_{b,j} + dis(p_a, p_b)$$

直接做是 $O(n^4)$ 的，但可以加一个中间步骤：

$$f_{a,j} = \min_b dp_{b,j} + dis(p_a, p_b)$$

$$dp_{i,j} = \min_a dp_{i,a} + f_{a,j}$$

这样就优化到 $O(n^3)$ 了。