浅谈线性基在 OI 中的应用

杭州第二中学 陈昕阳

摘 要

信息学竞赛中的线性基是解决一类子序列异或和问题的利器，且有着许多不同方向的扩展。本文先详细介绍了线性基最基础的构建方法与应用以及各种常见拓展，再在带删线性基这一方向提出了一些独到的方法与见解，解决了这一方向之前未能解决的一些问题。

1 导引

1.1 前言与定义

本文用 $\oplus$ 符号表示异或运算。

在 OI 中，一个由非负整数序列 S 的线性基通常指将这个序列内的所有元素视为二进制数（通常有较小的位数上界 m，即所有元素在 $[0, 2^m)$ 之间，本文中一般默认 $m \le 30$ 即对这些二进制数进行异或运算是 $O(1)$ 的），目标是构造另一个由 m 位二进制数序列 T，使得：

1. T 所有子序列（同时注意空子序列也是一种子序列）元素的异或和互不相同。
2. S 所有子序列元素的异或和**视为不可重集后**，与 T 所有子序列元素异或和构成的集合完全相同。
3. T 中所有元素最高位严格递减。

这样做的基本想法是借由性质二通过研究 T 子序列的异或和，来研究 S 的子序列异或和。而 T 因为满足这三点限制，具有更好的性质，研究 T 子序列异或和可能比研究 S 子序列异或和更简单。值得注意的是，这样的序列 T 并不唯一，存在很多不同但在应用上可能相互等价的选法。

实际上 OI 中的线性基与线性代数有很深的联系，此处谈论的 m 位二进制数可以等同为长度为 m 的系数在 $\mathbb{F}_2$ 中的向量，而要求构造的线性基就是给定的由 m 位系数在 $\mathbb{F}_2$ 中的向量构成的序列 S，张成的线性空间的一组基。不过这样忽略了限制三，再认为要求取出的基不允许有两个向量首个非零项值相同即可保持限制三。

而希望对一些向量张成的线性空间取一组基，方法当然是执行著名的高斯消元法，实际上这与后文介绍的线性基构建算法是完全等价的。这表明从线性代数的角度观 OI 中的线性基，其实在有相关知识的情况下会发现无非是给线性代数中的一些内容换了个名称，本质并无差别。

但考虑到大多数 OI 选手理应没有学习过线性代数相关知识，且实际上在大多数情况下引入线性代数的语言也并不能显著简化问题，只是徒增需要线性代数相关知识的门槛，本文会尽量避免使用线性代数的语言，而尽量使用虽然不甚严谨规范，但在 OI 中更广泛流传的语言来做描述。

1.2 导引

本文除本章引入与导引外，正文一共四章。其中前两章内容较为基础，也比较广为人知，有很多相关资料；后两章内容较为困难，且目前已有的介绍较少，其中最后一章内容均为原创，对多个现有的经典问题给出了复杂度更优秀的解法。

第二章会介绍线性基基本的构造方法，以及构造出线性基后的一些经典应用。

第三章则着眼于对于构建出线性基后容易解决的问题，如果每次都构造线性基时间开销太大该如何优化。即介绍了若干基于线性基性质的常见优化技巧。

第四章从更易懂的角度介绍了正交线性基，它是刻画点值序列以及执行线性基求交的强大工具。

第五章介绍了动态线性基，重点在于如何对线性基执行删除操作。介绍了本文原创的两种带删线性基，并展示了带删线性基与数据结构结合后可以发挥的强大功能。

2 构造算法与基本应用

2.1 基本性质观察

正如 1.1 所说，线性基最重要的思想是要研究 S 所有子序列异或和的性质，可以借由另一个具有更好性质，且所有子序列异或和构成的不可重集与 S 所有子序列异或和构成的不可重集完全相同的序列 T 来研究。为此，可以考虑研究对一个序列 S 做怎样的操作，其所有子序列异或和构成的**可重集**不发生改变，在这接下来的几节中这样的相等性质称为等价。

定理：从 S 内任意选出 $1 \le i \ne j \le |S|$，从 S 中删去 S_j 加入 $S_i \oplus S_j$，得到的序列 S' 与 S 等价。

证明：如果选出的子序列不包含 S_j，显然 S' 中存在对应的完全相同的子序列；如果选出的子序列同时包含 S_i, S_j，在 S' 中存在对应的子序列中删去 S_i, S_j 但加入了 $S_i \oplus S_j$ 的方案，异或和是相同的；如果选出的子序列包含 S_j 而不包含 S_i，S' 中存在对应子序列删去 S_i，

加入 $S_i \oplus S_j$ 以及 S_j，异或和不变。这给出了 S 子序列到 S' 子序列的对应方案，反方向的对应也容易构造，因此构建了子序列之间保持异或和不变的双射，显然具有等价性。

2.2 等价序列的增量构造法

有了基本的给序列内一个元素异或上序列中另一个元素，得到序列与原序列等价的性质，先介绍一个与常用的线性基的增量构造法略有不同的构造等价序列的增量构造法：

- 增量构造 $S'[1,i]$ 与 $S[1,i]$ 等价，且满足 S' 内 0 只出现在一段后缀，S' 内非零元素最高位单调严格递减。
- 初始 $i=0$，考虑将 i 增加 1 时如何从 $S'[1,i]$ 构造到 $S'[1,i+1]$，初始令 $S'_{i+1}=S_{i+1}$。
- 维护指针 j 初始指向 1，当 $j=i+1$ 时结束。否则考察若 S'_j 的最高位严格小于 S'_{i+1}，将 S'_{i+1} 插入到 S'_j 前面并结束将 i 增加 1 的增量构造；接下来若 S'_j 的最高位与 S'_{i+1} 相同，将 S'_{i+1} 异或上 S'_j 并将 j 增加 1；否则说明 S'_j 的最高位严格高于 S'_{i+1}，直接将 j 增加 1 即可。

可以发现这样确实达成了目标，略微需要注意的一点是当到达 $j=i+1$ 时的停止状态时，S'_{i+1} 是否为 0 都有可能。

S' 非零元素最高位单调递减，0 只出现在一段后缀的性质非常有趣。从 S' 中选取子序列考察异或和，异或和的值不与选出了多少个后缀 0 有关。而因为非零部分最高位单调递减，不难发现从非零部分任取子序列，得到的异或和都不相同。证明考虑任取两个不同的子序列，考察最靠前的出现于一个子序列而未出现于另一子序列的元素即可做到，两个子序列的异或和在该元素最高位上必然不相同。

这表明了假设 S' 中有 c 个 0，S' 所有子序列的异或和构成的可重集可以将看作是 S' 中非零元素所有子序列的异或和构成的不可重集复制 2^c 份拼成一个可重集。

这一结论将在后面被反复用到。

2.3 线性基的增量构造法

2.2 构造出的等价序列 S' 和在 1.1 中提及的目标序列 T 具有相仿、但是略微不同的性质。实际上注意到只要去掉 S' 中所有后导 0 就得到了一个可行的序列 T 即构造出了一组 S 的线性基。但注意在一些应用中实际上 S' 的后导 0 数量 c 也可能被用到，我们知道它代表了 S 子序列异或和构成的多重集每个元素的重复出现次数。

仿照等价序列的增量构造法，就得到了流传最广的线性基的增量构造法，这里介绍的版本记录了 c：

- 换一种方式描述等价序列。用值 c 记录后导 0 数量，对于 S' 中非零元素，假设其最高位是 p，将其的值记录在 w_p，不难发现不会产生冲突。如果 S' 中不存在非零元素最高位为 p，$w_p = 0$。

- 初始化 $c = 0$，$w_0 \sim w_{m-1}$ 均为 0。

- 增量构造加入 S_{i+1} 时，指针 j 初始指向 $m-1$。如果 S_{i+1} 不包含第 j 位，直接将 j 减 1，如果减完后 $j = -1$，将 c 加一并退出；否则如果 $w_j = 0$，将 w_j 设为 S_{i+1} 并退出；对于剩下的 $w_j \neq 0$ 的情况，将 S_{i+1} 异或上 w_j，再将 j 减 1，如果减完后 $j = -1$ 同理将 c 加 1 并退出。

可以发现这样只是换了一种方式描述了等价序列 S'，本质上没有区别。

增量加入一项 S_{i+1} 的时间复杂度为 $O(m)$，因此构造序列 S 的线性基的时间复杂度为 $O(|S|m)$，构造线性基的额外空间复杂度显然为 $O(m)$。

2.4 使用线性基求最大异或和子序列

假设已经对序列 S 得到了一组线性基 $w_0 \sim w_{m-1}$，根据 2.2 的性质求 S 的最大异或和子序列相当于求 $w_0 \sim w_{m-1}$ 的最大异或和子序列。

方便起见，将 $w_0 \sim w_{m-1}$ 中的所有非零元素按顺序为 $a_1 \sim a_k$。选择一个后缀 $a[p,k]$，考察两种 $a_p \sim a_k$ 的不同子序列 P,Q。根据等价序列的性质，P 与 Q 的异或和必然不相同（2.2 节）。更具体地说，P 与 Q 的异或和一定在某个不低于 a_p 最高位的位上不相同，而 $a_1 \sim a_{p-1}$ 的最高位都一定严格小于 a_p 的最高位，因此如果要将 P,Q 拓展为一组 $a_1 \sim a_k$ 的子序列，无论怎么拼上一组 $a_1 \sim a_{p-1}$ 的子序列，P 与 Q 中的异或和较小者都不可能反而变得异或和较大。

这表明要求 $a_p \sim a_k$ 的最大异或和子序列，只需要先求出 $a_{p+1} \sim a_k$ 的最大异或和子序列，再考虑要不要额外选上 a_p 这一项元素即可。

算法流程可以简化为：

- 维护最优答案 ans 初始为 0，指针 j 初始指向 $m-1$。

- 只要 $j \geq 0$，将 ans 设为 $\max(ans, ans \oplus w_j)$，然后将 j 减一。

最后的 ans 即为目标答案，该算法时间复杂度为 $O(m)$。

可以发现即使 ans 的初始值并非 0 而是某个给出的值 v，即要求的是异或和再异或上给定值 v 后，得到权值最大的子序列的权值，上述的分析也仍然完全可行，因而算法也仍然正确。

2.5 使用线性基求异或和为某个定值的子序列个数

根据 2.2 的性质，如果 S 序列存在子序列异或和为目标值，子序列个数为 2^c，否则为 0。

而要判断 S 序列是否有子序列异或和为目标值 x，只需要尝试向 S 的线性基内插入 x：如果插入失败表明向 S 序列加入 x 后，其所有子序列异或和构成的不可重集没有扩大，因此 x 本来就作为 S 内某个子序列异或和出现；如果插入成功，同理表示不可重集扩大了，因此 x 并未作为 S 内某个子序列异或和出现。

该算法时间复杂度为 $O(m)$。

实际上该问题可视作 2.6 的一个特例。

2.6 使用线性基求异或和小于某个定值的子序列个数

仍然利用 2.2 的性质，将问题转化为求 w 所有非零项的异或和小于某个定值 lim 的子序列个数，最后将答案乘上 2^c。设 w 非零项依次为 $a_1 \sim a_k$。

仍然考察后缀 $a[p,k]$，考虑取出的 a 的子序列在 $a[p,k]$ 内的部分的异或和 cur，将 cur 和 lim 的不低于 a_p 最高位的所有位做比较：

- cur 较大，$a_1 \sim a_{p-1}$ 最高位都严格小于 a_p 的最高位，即使考虑它们也无法改变 cur 在某个高位超过 lim 的事实。
- lim 较大，同理任意选择 $a_1 \sim a_{p-1}$ 的一个子序列都合法。
- cur 与 lim 在这些位上完全相同，那么可以认为目标变为数 $a_1 \sim a_{p-1}$ 有多少个子序列异或和异或上 cur 低于 a_p 最高位的部分的结果，小于 lim 低于 a_p 最高位的部分，递归到更小的子问题。

由此导出了以下算法：

- 维护当前异或和 cur 初始为 0，当前答案 ans 初始也为 0，指针 j 初始指向 $m-1$，重复执行以下操作直到 $j<0$ 或退出：
- 如果 lim 在第 j 位上为 1
  - $w_j \neq 0$，先将 ans 加上 $2^{\sum_{k=0}^{j-1}[w_k\neq 0]}$。如果 cur 在第 j 位上为 0，将 cur 异或上 w_j。无论如何将 j 减 1。
  - $w_j = 0$，如果 cur 在第 j 位上为 0 将答案加上 $2^{\sum_{k=0}^{j-1}[w_k\neq 0]}$ 并退出，否则将 j 减 1。
- 如果 lim 在第 j 位上为 0
  - $w_j \neq 0$，如果 cur 在第 j 位上为 1，将 cur 异或上 w_j。无论如何将 j 减 1。

– $w_j = 0$，如果 cur 在第 j 位上为 1 则退出，否则将 j 减 1。

- 最后将 ans 乘上 2^c 作为答案。

这些情形的分类讨论无非是基于总可以剪枝掉选择 w 非零项一段后缀的子序列时，选出的异或和与限制在已经考虑的高位上并非完全相同的情况，只留下两者完全相同的情况继续递归考虑。

该算法的时间复杂度在合理预处理后为 $O(m)$。

对这个算法稍加修改，利用同样的思想即可在 $O(m)$ 时间内求出异或和第 k 小的子序列的异或和值，在此不详细展开。

2.7 利用线性基解决无向图上最大 XOR 和路径问题

2.4 ~ 2.6 的应用都局限于研究序列上子序列异或和相关问题，在这一章的最后，我们来看一个更加有趣的应用：

例题 2.1. [1]

给定一张边带权的无向连通图 $G = (V, E)$，边权在 $[0, 2^m)$ 之间。求所有从 1 到 n 的路径（不要求简单）中，经过的所有边边权异或和最大的一条的边权异或和，注意一条边在异或和中被计算的次数是它被经过的次数。

$|V|, |E| \le 10^5, m \le 60$。

这个问题当然与无向带权图上的最短路问题非常不同，这是因为一条路径再拓展一步，即额外加上一条边之后，边权异或和可能减小也可能增大，类似 Dijsktra 的分析完全失效。

一个观察是，对于同一条路径正反走两次对异或和不会产生任何影响，虽然走完之后所在节点也不会发生变化。但其实可以任选一个走过路径上的节点，再选择一条包含这个节点的回路，在正反走这条路径中首次经过这个节点时，将回路插入到剩余的需要经过的边前面，就将异或和异或上了该回路的边权异或和。

而给定的图是无向连通图，因此任取一个回路，总能走到回路上的任意一个点再走回来，因此假设当前找到了一条 $1 \to n$ 的路径权值异或和为 val，存在一个图上的回路边权异或和为 x，总能找到一条权值为 $val \oplus x$ 的 $1 \to n$ 的路径。

定理：将无向图上所有回路边权异或和去重后记录为序列 S，任取一条 $1 \to n$ 的路径边权异或和为 v，$1 \to n$ 的最大异或和路径权值为 S 中任选子序列，子序列内元素异或和异或上 v 的最大值。

证明：设这样算出的答案为 ans'，真实答案为 ans。$ans \ge ans'$ 是显然的，因为确实可以由 v 对应的路径拼上 S 内某个子序列对应的回路找到一条权值为 ans' 的 $1 \to n$ 路径。$ans \le ans'$ 则是考虑，先走最优路径再沿着给出的 $1 \to n$ 路径返回（即倒着走，无向图因此

[1] 来源：[WC2011] 最大 XOR 和路径 https://www.luogu.com.cn/problem/P4151

合法），再沿着给出的 $1 \to n$ 路径走到 n，前两部分构成一条回路，因此权值为 ans 的方案被考虑到了。

于是问题转化为求出 S 序列的任意线性基即可，但按照定义求解需要枚举无向图上的所有回路，这有无穷多条，当然不可行。但注意到只需要求出 S 序列的线性基，实际上可以只考察很少的回路：

定理：考虑任意生成树，只考虑所有非树边 (u, v) 加上 u 到 v 简单路径构成的回路，边权异或和构成的线性基与原图所有回路构成的线性基等价。

证明：显然这样求出的线性基 $basis'$ 不可能比原图所有回路边权异或和构成的线性基 $basis$ 大，于是只需要证不存在某个原图上的回路，其异或和不能写成 $basis'$ 内一些元素的异或和（从而将其插入 $basis'$ 会扩大 $basis'$），这只需要真的给出一组写成 $basis'$ 内一些元素异或和的方案。考虑任取根 r，记生成树上 r 到 u 简单路径边权异或和为 d_u，任意回路边权异或和也可以表示为对其包含的所有边（经过多次计入多次）(u_i, v_i, w_i) 将 $d_{u_i} \oplus d_{v_i} \oplus w_i$ 全部异或起来后得到的结果。对于树边，结果为 0，对于非树边，结果这条非树边所对应的回路（被考虑过的）的边权异或和。

于是只需要先 $O(|V| + |E|)$ 求出任意生成树并找到任意一条 $1 \to n$ 的路径求出边权异或和。然后将 $O(|E|)$ 条非树边对应的回路的边权异或和插入线性基，再最后做一次 2.4 的查询即可，时间复杂度为 $O(|V| + |E|m + m)$。

2.8 本章小结

本章先指出了线性基原理中最重要的基本性质，利用基本性质立即推导出了等价序列的增量构造法，并指出最常用的线性基的增量构造法只是给等价序列的增量构造法换了一种写法。

接下来 2.4 ~ 2.6 展示了线性基最基本的三个应用，实际上这三个应用基本的想法都是一致的：利用线性基内元素最高位严格递减的特性，在某个位处考虑选出的方案在不低于这个位的部分的权值，而这个权值不受线性基内最高位更低的元素的影响，由此可以直接减枝掉很多局面。利用这一基本思想还可以解决更多更复杂的问题。

2.7 是非常经典的图上最大 XOR 和路径问题，本文给出做法并详细论证了做法的正确性。

3 常见优化技巧

3.1 线性基合并

考虑这样一个问题：假设现在已知序列 S 的一组线性基是 S'，序列 T 的一组线性基是 T'。现在要将序列 S 和序列 T 拼接在一起，能否在只知道 S', T' 的情况下求出 $S + T$（表示

序列拼接）的线性基？

从线性基实际上是等价序列去掉所有后导 0 的这一点来看，做法是显然的：直接将 S', T' 作为序列拼接起来，求这个序列的线性基得到的就是 $S+T$ 的线性基。这是因为 S' 拼上若干个 0 后与 S 等价，T' 拼上若干个 0 后与 T 等价，那么显然 $S'+T'$ 再拼上若干个 0 与 $S+T$ 等价，于是直接对 $S'+T'$ 构造等价序列即可。

如果额外记录了 c 值，合并起来后的 c 值也是容易计算的。

一次线性基合并需要将 $O(m)$ 个元素插入线性基，因此时间复杂度为 $O(m^2)$。

3.2 结合数据结构

针对序列 S 构造的线性基（在不加下文所介绍的其他的优化前），只能处理类似询问从整个序列中任取一个子序列，子序列权值异或和最大是多少之类的问题，并且还不支持修改。如果要面对区间询问或带修，结合数据结构是一种可行的思路。

例题 3.1. [2]

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a。支持 q 次操作分为两种：

- 修改操作：将某个 a_x 修改为 y，保证 $y \in [0, 2^m)$。
- 查询操作：给出区间 $[l, r]$，查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \le 5 \times 10^4, m \le 30$。

考虑对原序列建立线段树，在对应区间为 $[l, r]$ 的节点维护 $a[l, r]$ 构成的任意一组线性基。注意到线性基可以 $O(m^2)$ 合并，于是可以简单的把一个序列的线性基看作是一种满足结合律的信息，用 $O(n + q \log n)$ 次信息合并来保证每次询问时能顺利求出 $a[l, r]$ 的任意一组线性基，进而回答询问。

该算法的时间复杂度为 $O((n+q\log n)m^2)$，实际上可以分析到略好的上界但在此不赘述。

可以发现这个算法的时间复杂度并不优秀，本章后面几节会介绍在与这个问题相似的一些更为简单的问题上该如何优化复杂度，在第五章会真正介绍如何优化这个问题的时间复杂度降低至 $O(nm + qm(\log n + m))$。

3.3 维护等价序列优化复杂度

如果要维护序列 S 的线性基但这难以做到，可以考虑维护与 S 等价的序列构成的线性基。这在静态的时候当然没有帮助，但如果要针对序列 S 进行一些修改，有时会很有效果。

[2] 来源：经典问题, 出处不明

例题 3.2. [3]

维护一个长度为 n 值域在 $[0,2^m)$ 之间的非负整数序列 a。支持 q 次操作分为两种：

- 修改操作：给出区间 $[l,r]$ 和值 $v\in[0,2^m)$，将 $a_l\sim a_r$ 全部异或上 v。
- 查询操作：给出区间 $[l,r]$，查询从 a 的子区间 $a[l,r]$ 中任取一个子序列能得到的异或和最大值。

$n,q\le 5\times 10^4, m\le 30$。

设 $b_i=a_i\oplus a_{i-1}$，可以发现 $a[l,r]$ 等价于 $a_l+b[l+1,r]$。而对 $a_l\sim a_r$ 全部异或上 v 的操作对 b 的影响只是将 b_l,b_{r+1} 异或上了 v，可以视作是对 b 进行两次单点修改。于是用线段树维护 b 序列做 3.2 的问题每次查询出 $b[l+1,r]$ 构成的任意线性基，再将 a_l 插入该线性基即可解决此问题，需要略微注意的一点是还需要支持对 a 进行区间异或，单点查询，但这也不难做到，时间复杂度仍为 $O((n+q\log n)m^2)$。

注意不要将此题视作孤例，如果希望以“维护等价序列”作为做法命题，很容易就能得到以下题目：

例题 3.3. [4]

维护一个长度为 n 值域在 $[0,2^m)$ 之间的非负整数序列 a，给定一个常数 k。支持 q 次操作分为两种：

- 修改操作：给出区间 $[l,r]$ 和值 $v\in[0,2^m)$，对于 $\forall l\le i\le r$ 且满足 $i \bmod k = l \bmod k$，将 a_i 异或上 v。
- 查询操作：给出区间 $[l,r]$，查询从 a 的子区间 $a[l,r]$ 中任取一个子序列能得到的异或和最大值。

$n,q\le 5\times 10^4, k,m\le 30$。

做法大同小异，维护 $b_i=a_i\oplus a_{i-k}$ 即可，时间复杂度为 $O((n+q\log n)m^2+qkm)$。实际上如果是给 $[l,r]$ 内编号模 k 在某个每次修改给出的集合 S 中的下标做异或 v 的操作，时间复杂度也可以做到 $O((n+q\log n+qk)m^2)$。

[3]来源：[Ynoi2013] 无力回天 NOI2017 https://www.luogu.com.cn/problem/P5607

[4]来源：原创题

3.4 前缀线性基

前缀线性基的想法可以从两个动机看出。

1. 最常用的构造序列 S 的线性基的方法是增量构造法，每次考察 S 的首个尚未被考虑过的元素，考虑加入其对线性基的影响，而我们知道要么没有影响，要么只是将某一项值由 0 填成某个非零值。所以如果记录了每个非零值是在哪一次插入时被填上的，实际上自然就可以得到 S 所有前缀的线性基信息了：$S[1,p]$ 的线性基信息就等于 S 整体的线性基信息，只保留被填上时间不晚于 p 的非零值。

2. 实际上对 S 任意重排后，建出的线性基都彼此等价。因此可以给 S 内的每个值额外分配一个优先级，钦定按照优先级从高到低的顺序增量构造，将值插入线性基。

现在考虑这样一个问题：需要动态维护 (t_i,a_i) 二元组集合 S，你需要在每次向 S 内额外插入一个二元组后，求出 $f(S)$ 表示把所有二元组按照优先级 t_i 从大到小排序后，依次将 a_i 插入到线性基，线性基的形态。保证 t_i 互不相同。

我们先给出以下在 2.2 中描述的算法基础上稍加修改后的算法，再说明其正确性：

- 在维护 w_i 的基础上记录 p_i 表示占领第 i 位的二元组的优先级，初始均为 $-\infty$。

- 向 S 内插入 (t_i,a_i) 时，指针 j 初始指向 $m-1$。如果 a_i 不包含第 j 位，直接将 j 减 1，如果减完后 $j=-1$，将 c 加一并退出。否则如果 $p_j<t_i$，交换 w_j,a_i，再交换 p_j,t_i。无论如何，将 a_i 异或上 w_j，再将 j 减 1，如果减完后 $j=-1$ 将 c 加 1 并退出。

正确性证明：交换相当于保证了一定优先级较高的二元组占领第 i 位，将优先级较低的二元组的 a 异或上优先级较高的在“按照优先级降序插入”的题设下显然也确保了等价性。

该算法的时间复杂度仍为 $O(m)$。

3.5 前缀线性基的多种应用

前缀线性基应用广泛，在此节暂且用以下三个例题说明，后文还有和其他技巧结合的例题：

例题 3.4. [5]

给定长度为 n 值域为 $[0,2^m)$ 的非负整数序列 a。q 次询问区间 $[l,r]$ 查询从 a 的子区间 $a[l,r]$ 中任取一个子序列能得到的异或和最大值。

$n,q\le 5\times 10^5, m\le 30$。

[5]来源：CF1100F Ivan and Burgers https://codeforces.com/problemset/problem/1100/F

这是 3.2 例题无修改的弱化版，考虑要回答询问 $[l,r]$，只需要将 $\forall 1 \le i \le r,(i,a_i)$ 插入集合 S 然后保留只 $f(S)$ 中 $p_i \ge l$ 的部分就得到了 $a[l,r]$ 的线性基信息。而对 (i,a_i) 二元组序列的每个前缀求其 f 显然就是上面描述的问题，因此得到了 $O((n+q)m)$ 的算法。

例题 3.5. [6]

给定 n 个点每个点带 $[0,2^m)$ 内非负整数点权的树 T。q 次询问 u,v 查询 u 到 v 简单路径上所有点点权构成的序列中任取一个子序列能得到的异或和最大值。

$n \le 5\times 10^5, q \le 10^5, m \le 30$。

这是上一个例题上树的情形。任意定根 r，先考虑 u,v 有祖先后代关系时该如何获取 u 到 v 简单路径上点权构成序列的线性基信息。不妨设 u 是 v 的祖先，可以发现将 v 祖先中所有点 w 的 (dep_w,a_w) 插入集合 S 然后只保留 $f(S)$ 中 $p_i \ge dep_u$ 的部分就得到了目标信息，其中 dep_p 表示节点 p 到根简单路径边数。注意到动态维护 $f(S)$ 的是一个增量算法，且该算法的时间复杂度不依赖于均摊，因此从一个局面增量到不同局面的时间复杂度仍然正确（也就是可以把一个点的信息复制给它的所有子节点，再插入它子节点对应的那个二元组，也仍然完全可行）。因此该子问题仍然可以 $O((n+q)m)$ 解决。

对于不存在祖先后代关系的情形，只需要求一次 LCA 将两条有祖先后代关系的链的线性基信息 $O(m^2)$ 合并即可。

因此时间复杂度为 $O(nm+qm^2)$，LCA 可能有额外时间开销但这里不关心。

例题 3.6. [7]

维护元素在 $[0,2^m)$ 内的非负整数集合 S，支持插入、删除元素，每次操作后求选出一个子集异或和的最大值。

操作次数 $q \le 5\times 10^5$，$m \le 30$。

注意到本题允许离线，因此可以求出每次插入 a_i 的操作中，a_i 在第 t_i 次操作被删除，不存在则为 $+\infty$。那么进行前 q' 次操作后的线性基信息可以看作是将前 q' 次操作中的所有插入操作看作二元组 (t_j,a_j) 插入集合 S，询问 $f(S)$ 中 $p_i > q'$ 的部分。

因此该问题也在 $O(qm)$ 的时间内得以解决。

本节的第三个例题表明实际上使用前缀线性基的技巧就能一定程度上实现动态线性基，但实际上这种实现方法是非常“脆弱”的。首先这个算法是离线的，难以优化到在线。另外为了实现这种虚假的在线，这一算法还已经“消耗”掉了线性基内可以控制优先级的自由元，这导致就算不要求强制在线，这个算法也不可能做到对一个序列单点修改，支持查询其任意前缀的线性基信息。在第五章会介绍真正的动态线性基，然而这一算法就算不考虑其复杂性，也仍然有其自身的缺陷。

[6]来源：SCOI2016 幸运数字 https://www.luogu.com.cn/problem/P3292

[7]来源：经典问题, 出处不明

3.6 线性基的进一步消元

在建出线性基 $w_0 \sim w_{m-1}$ 之后，其实还可以进行进一步消元以保证若 $w_i > 0$，所有 $\forall i < j \le m-1$ 的 w_j 不包含第 i 位。

做法是简单的：升序枚举 $i \in [0,m)$，若 $w_i > 0$，再升序枚举 $j \in (i,m)$，若 w_j 含有第 i 位，将 w_j 异或上 w_i。

该做法的正确性可以归纳证明：假设在前 p 轮结束后，保证了 $\forall 0 \le i \le p$ 的位 i 满足这一性质。那么若 $w_{p+1} > 0$，第 $p+1$ 轮无疑保证了第 $p+1$ 位也满足这一性质，且没有破坏 $[0,p]$ 这些位的性质。

做这样的进一步消元有时并不必要，但有时很有帮助，可以通过以下例题看出：

例题 3.7. [8]

维护一个长度为 n 值域在 $[0,2^m)$ 内的非负整数序列 a 以及一个元素同样是 $[0,2^m)$ 内非负整数的集合 S_0。初始 a 给定 S_0 为空集，支持 q 次操作分为三种：

- 对 a 的修改操作：给出 x,v，将 a_x 异或上 v，v 仍然 $\in [0,2^m)$。
- 对 S_0 的修改操作：给出 v，将 v 插入 S_0。
- 查询操作：定义 $g(x,S) = \max\limits_{T \subseteq S}(x \oplus \mathrm{xor}(T))$，其中 $\mathrm{xor}(T)$ 表示 T 集合内所有数的异或和。给出 $[l,r]$ 求 $\sum\limits_{i=l}^{r} f(\oplus_{j=l}^{i} a_j, S_0)$，答案对 2^{64} 取模。

$n \le 5 \times 10^5, q \le 10^5, m \le 64$。

先假设 S_0 是某个一开始就给定的集合，考虑如何回答查询操作。首先对 S_0 建线性基并执行上述的进一步消元，然后我们考虑如果询问的是 $g'(x,S) = \min\limits_{T \subseteq S}(x \oplus \mathrm{xor}(T))$ 该如何处理：实际上由于对于某个 $w_i > 0$ 的 i，由于线性基内除了 w_i 之外不存在其他数包含位 i，那么只要 x 包含位 i，在最小异或值方案中就必须选 w_i，否则就一定不选，不然一定不优。由此便可看出最小异或值方案是唯一的使得所有 $w_i > 0$ 的位 i 一定在异或和中不被包含的选子集的方案。注意到每次用到的 x 实际上是若干数 $x_1 \sim x_k$ 的异或和，如果对每个 x_i 都求出 $g'(x_i,S)$ 并将所有 $g'(x_i,S)$ 异或起来，其对应的选取子集的方式就是这组方案。即我们证明了 $g'(x,S) = \oplus_{i=1}^{k} g'(x_i,S)$。

但实际上需要研究的是 $g(x,S)$，不过实际上可以注意到 $g(x,S) \oplus g'(x,S) = \oplus_{i=0}^{m-1} w_i$。这是因为最大异或值方案反过来是使得所有 $w_i > 0$ 的位 i 一定在异或和中被包含的取子集的方案，所以对于同一个 w_j，其一定恰在最大异或和方案与最小异或和方案中被取一次。

于是询问可以这样回答：维护 $b_i = g'(a_i,S_0)$，$\sum\limits_{i=l}^{r}(\oplus_{j=l}^{i} b_j) \oplus bas$ 即为答案，其中 bas 表示 $\oplus_{i=0}^{m-1} w_i$。

[8] 来源：2023-2024 集训队互测 Xor Master https://qoj.ac/contest/1417/problem/7765

回到原问题，对 a 做单点修改操作相当于单点修改 b，花费 $O(m)$ 时间即可求出修改后的结果。对 S_0 做修改操作如果没有扩大线性基，所有 b 均不变，否则一个暴力的想法是直接重新 $O(nm)$ 求出 $b_1 \sim b_n$，这种事件只会发生 m 次。问题可以转化为维护一个长度为 n 的序列，有 m 次整体重建，q 次单点修改以及查询一个区间所有前缀的异或和再异或上给定值 C。该部分做法与本文主题无关但可以在 $O(nm + q\log^2 n)$ 内解决。

但这样只能得到 $O(nm^2 + q\log^2 n)$ 的解法，实际因为 $O(nm^2)$ 太大无法通过。

注意到瓶颈在于 $O(m)$ 次在 S_0 对应线性基扩大时直接重新计算了所有 $b_1 \sim b_n$，实际上执行线性基的进一步消元之后，求 $g'(x, S_0)$ 不必再按照从高位到低位的顺序将结果向额外异或上线性基内元素的方案取 min，而是任意顺序都可行。同时如果线性基扩大导致线性基内出现了最高位为第 p 位的元素，注意到实际上知道线性基扩大前的 b_j，而 b_j 在线性基内其他 $w_i > 0$ 的 i 位上一定均为 0，于是只需要检查 b_j 是否包含第 p 位即可。这样重新计算 $b_1 \sim b_n$ 的时间复杂度降为 $O(n)$。线性基扩大后再次执行进一步消元也只需要试着消掉 w_p 的更低位，并用 w_p 去消掉 $\forall p < k < m$ 的 w_k 的第 p 位，于是进一步消元也是 $O(m)$ 的。

于是该做法的时间复杂度被优化到了 $O(nm + q\log^2 n + m^2)$ 。

实际上 $g'(x, S), g(x, S)$ 之间的结论不用进一步消元的方法也能证明，但做最后的时间复杂度的优化，进一步消元是必不可少的。

3.7 前缀线性基结合数据结构

让我们考虑 3.2 的另一个相仿的问题。

例题 3.8. [9]

维护 n 个序列 $S_1 \sim S_n$，初始均为空序列。支持 q 次操作分为两种：

- 插入操作：向 S_x 的末尾插入 y，保证 $y \in [0, 2^m)$。
- 查询操作：给出区间 $[l, r]$，查询将 $S_l \sim S_r$ 这些序列全部按顺序拼接起来后，从中任取一个子序列能得到的异或和最大值。

$n, q \le 2 \times 10^5, m \le 30$。

直接沿用 3.2 的做法当然并没有什么问题，但实际上利用前缀线性基的性质，可以做得更好：

建线段树，对每个线段树上的节点，如果它是一个左儿子，维护对应区间的后缀线性基；如果它是一个右儿子，维护对应区间的前缀线性基；对根什么都不维护。

查询时，考虑何时查询区间首次被 mid 切开，设此时线段树节点管辖 $[l, r]$，查询区间是 $[L, R]$，需要的线性基信息可以由 $[L, mid], [mid + 1, R]$ 做一次 $O(m^2)$ 的合并得到，而这两段

[9] 来源：P 哥的桶 https://www.luogu.com.cn/problem/P4839

区间分别是左儿子对应区间的后缀和右儿子对应区间的前缀，信息都可以通过维护的前/后缀线性基 $O(m)$ 提取出来。找查询区间首次被 *mid* 切开的位置其实是求一次 LCA，理论上不是瓶颈。

修改时，对所有管辖被修改位置的线段树上节点，向这个节点的前/后缀线性基做一次插入即可。

这个算法相比 3.2 的做法，单次修改复杂度降为 $O(m \log n)$，单次查询复杂度降为 $O(m^2)$。

该例题又一次证明了前缀线性基的强大功能，但要使用前缀线性基，得先拆出“前缀询问”的形式。

但如果修改并非向某个序列末尾插入元素，而是修改某个序列中的某个元素（即 3.2 中的问题）。如果还想使用类似的方法，会需要解决 3.5 中提到的“对一个序列单点修改，支持查询其任意前缀的线性基信息”这样的问题，而这一问题在目前的讨论中，还暂时没有较好的方法。

3.8 本章小结

本章主要聚焦于对于需要用线性基解决的题目，如何快速求出需要的线性基信息。3.1 ~ 3.6 介绍了线性基合并、使用数据结构维护信息、转而维护更易维护的等价序列、前缀线性基、对线性基进行进一步消元这些常用技巧。这些常用技巧有时需要配合使用，这正是 3.7 的内容。

3.2 使用数据结构维护信息的方法基于 3.1 的线性基合并。这一方法能力非常强，但线性基合并单次 $O(m^2)$ 的代价太高往往难以接受。3.5 和 3.7 描述了如何使用 3.4 介绍的前缀线性基的技巧尽量少用或不用线性基合并来降低复杂度，这一优化方式非常常见。

3.3 描述的维护等价序列的方法是一种转化来简化问题的手法，在转化完之后仍然需要使用剩下几节的方法维护。

3.6 对线性基进行进一步消元的技巧在一些情况下也可以优化时间复杂度，但相对不太常用。

本章最后遗留下来的问题是，如何延续 3.7 中的思路优化 3.2 中的问题，这将在第五章得到解答。

4 正交线性基

4.1 准备工作

事实上正交线性基与线性代数的联系就更为紧密了，但避免从线性代数的角度介绍仍然是可行的，为此需要先介绍一些概念，但它们在 OI 中仍然是为人熟知的。

本章会区分原序列 S（不定长）和一些需要用到的，定长为 2^m 描述子序列异或和分布信息的序列，称后者为点值序列，虽然有时用到的点值序列可能找不到其对应的原序列是什么。

需要用到下面四个定义：

- 序列 S 对应的点值序列：对于某个序列 S，定义其对应的点值序列是长度为 2^m，编号在 $[0,2^m)$ 中的序列 $v(S)$，其中 $v(S)_i$ 表示 S 子序列中异或和为 i 的子序列数量。
- 点值序列的异或卷积：对于两个点值序列 a,b，定义它们的异或卷积 $a\times b$ 结果也是一个点值序列 c，满足 $c_k=\sum\limits_{0\le i,j<2^m,i\oplus j=k}a_ib_j$。
- 点乘、正交：对于两个 m 位二进制数 i,j，定义 $i\odot j=\text{popcnt}(i\&j)\bmod 2$，如果 $i\odot j=0$ 称 i 和 j 正交。
- FWT 变换：对于一个点值序列 a，定义对其进行 FWT 变换后的结果 $\hat{a}$ 是一个点值序列，满足 $\hat{a}_i=\sum\limits_{j=0}^{2^m-1}(-1)^{i\odot j}a_j$。

可以发现按照异或卷积的定义，事实上对任意原序列 S，有 $v(S)=\prod\limits_{i=1}^{|S|}v(S_i)$，这里连乘记号指异或卷积。以下针对点值序列整体的连乘记号均指异或卷积，如果取出点值序列具体某一项进行连乘，当然就是正常乘法。

容易验证点乘与异或运算满足分配律即 $i\odot(j\oplus k)=(i\odot j)\oplus(i\odot k)$。

对于 FWT 变换，需要知道对 $\hat{a}$ 再进行一次 FWT 得到的结果是给 a 每一项系数乘上 2^m。以及如果 $c=a\times b$，那么 $\hat{c}$ 等于 $\hat{a}$ 和 $\hat{b}$ 逐点相乘，下面记 p,q 两个点值序列逐点相乘的结果为 $p\cdot q$。这两点性质的证明在此略去。

4.2 点值序列 FWT 变换后的结果

现在我们来研究对于序列 S，其点值序列 $v(S)$ 进行 FWT 后得到的 $\hat{v}(S)$ 的第 k 项的值是什么。

那么由于 $v(S)=\prod\limits_{i=1}^{|S|}v(\{a_i\})$，根据 FWT 的性质 $\hat{v}(S)_k=\prod\limits_{i=1}^{|S|}\hat{v}(\{a_i\})_k$，而 $\hat{v}(\{a_i\})_k=(-1)^{0\odot k}+(-1)^{a_i\odot k}=1+(-1)^{a_i\odot k}$。因此当且仅当 $\forall 1\le i\le|S|,a_i\odot k=0$，此时这个连乘积项非零，值为 $2^{|S|}$，剩下的情况 $\hat{v}(S)_k$ 都必然为 0。

也就是说，$\hat{v}(S)_k$ 在 k 与 S 内所有元素正交时为 $2^{|S|}$，否则为零。

这为研究点值序列提供了另一种角度：如果能找出所有的 k 使得 k 与 S 内所有元素正交，S 的点值序列 $v(S)$ 就是将所有这些 k 的位置值设为 $2^{|S|}$，剩余位置设为 0 得到的点值序列进行一次 FWT 后得到的点值序列，每项除以 2^m。

而要研究 k 是否与 S 内所有元素正交，还有以下非常好用的结论：$k\odot x=0$ 且 $k\odot y=0$ 充要于 $k\odot x=0$ 且 $k\odot(x\oplus y)=0$，这用点乘对异或的分配性即可证明。因此研究 k 是否与

S 内所有元素正交，等价于研究 k 是否与 S 的任意等价序列正交，等价于 k 研究 k 是否与 S 的任意线性基内所有元素正交。

新角度的好处在于，一些情况下与所有 S 内元素正交的 k 非常少，从而这样构造出的点值序列非常稀疏，而稀疏性是非常有用的。最为简单的例子就是通过 S 构造出的线性基为满线性基的情况，此时如果对线性基 3.6 提到的进行进一步消元将得到 $w_i = 2^i$ 的线性基，此时 k 与所有 2^i 正交，不难发现 k 只能为 0。

可以通过以下例子来看出新角度非常有用：

例题 4.1. [10]

给定长为 n 值域为 $[0, 2^m)$ 的非负整数序列 a，对 $\forall 0 \le i \le m$ 求其有多少个子序列异或和的 popcnt 为 i。

$n \le 2 \times 10^5, m \le 53$。

当然只需要在 a 的任意一组线性基上考虑原问题，最后将答案每项乘以 2^c 即可。记 a 线性基内非零元素有 s 个，对于 $s \le \frac{m}{2}$ 时可以直接枚举线性基的所有子序列共 2^s 种，接下来考虑 s 较大的情况。

那么要研究的无非是线性基对应的点值序列在 popcnt 上的分布，使用新角度把点值序列视作与线性基内所有元素正交的 k 位置上为 2^s，剩余位置为 0 的点值序列 FWT 后除以 2^m 的结果。可以发现其实只需要知道所有 occ_p 表示与线性基内所有元素正交的 k 中，popcnt 为 p 的恰好有 occ_p 个，实际上就已经可以推导出答案。

这是因为最后要求的是这些 k 构造出的点值序列 FWT 后得到的新点值序列在 popcnt 上的分布，那么对于一个 $\text{popcnt}(k) = p$ 的 k，考虑其对于所有其为 1 的 p 个位上有 a 个位置也为 1，其为 0 的 $m - p$ 个位上有 b 个位置为 1 的所有 k'，$(-1)^{k \odot k'}$ 的贡献之和，可以发现这样的 k' 有 $\binom{p}{a}\binom{m-p}{b}$ 种，且 $(-1)^{k \odot k'}$ 固定为 $(-1)^a$，并且这些 k' 每个的 popcnt 均为 $a + b$，所以可以看作是它为 ans_{a+b} 这一项提供了 $\frac{\binom{p}{a}\binom{m-p}{b}(-1)^a}{2^m}$ 的贡献。因此在求出所有 occ_p 之后，可以 $O(m^3)$ 解决原问题。

直觉上来说，当 s 较大，与线性基内所有 s 个元素同时正交的 k 会比较少，因此暴力枚举所有这些 k 的时间复杂度应该还可以接受，但问题在于，如何快速找出所有这些 k？这就需要使用正交线性基来解决了。

4.3 正交线性基的构造

延续 4.2 的内容，先对 a 已有的这组线性基进行进一步消元，接下来我们声称：对于所有 $w_i = 0$ 的位 i 任意选取 0 或 1，这样一共 2^{m-s} 种方案，每种方案恰好对应一种将所有 $w_i > 0$ 的位填上 0 或 1 的方案，得到一个数 k 与线性基内所有元素全部正交。

[10] 来源：CF1336E2 Chiori and Doll Picking https://codeforces.com/problemset/problem/1336/E2

假设 k 的所有 $w_j = 0$ 的位已经全部选好。考虑某个 $w_i > 0$ 的位，因为 k 要与 w_i 正交，且 w_i 满足其不包含除自己以外的任何 $w_j > 0$ 的位 j，所以要判断 k 与 w_i 是否正交，现在唯一未知的项就是 k 在位 i 上的取值，取值方案显然只有一种：如果 k 在 $w_j = 0$ 的位上与 w_i 有奇数个位置重合，那么 k 必须包含位 i，否则必须不包含位 i。

这也可以看作是，对每个 $w_j = 0$ 的位 j，如果 k 选上了位 j，且 $j < i < m$ 的 w_i 也包含位 j，每多一个这样的位 j，k 是否包含位 i 的包含性会发生一次翻转。

既然这样，我们可以写下一个编号为 $0 \sim m-1$ 值域为 $[0, 2^m)$ 的序列 w'，$w'_i \neq 0$ 当且仅当 $w_i = 0$，且 $w_i = 0$ 时，$w'_i = 2^i + \sum_{j=i+1}^{m-1} [w_j \& 2^i = 2^i] 2^j$。如果希望 k 包含某个子集的 $w_i = 0$ 的位 i，最终对应的与线性基内所有元素正交的 k 就是这个子集的位 i 对应的 w'_i 值的异或和。

这样的 w' 被称为 a 对应线性基的正交线性基，它的特点是其所有子序列异或和构成的集合，恰好等于与 a 线性基内所有元素正交的 k 构成的集合。

如果已知 a 的一组线性基，构造其正交线性基的复杂度当然不会超过 $O(m^2)$，更精细的实现可以做到 $O(sm)$，其中 s 还是表示 a 的这组线性基的非零元素个数。

需要格外注意的一点是，构建正交线性基前必须对 a 的线性基进行进一步消元，否则求出的结果可能不正确。

于是 4.2 中的问题也得到了完美解答：只需构造出 a 对应线性基的正交线性基，枚举正交线性基的所有子序列并求出异或和，即可快速找出所有的 k，这样做的时间复杂度为 $O(2^{m-s})$，平衡两部分做法可以在 $O(nm + 2^{\frac{m}{2}} + m^3)$ 时间复杂度内解决原问题。

4.4 线性基求交

在一些题目中，需要快速判断多个序列是否均有子序列异或和为某个给定值 x，此时需要所谓“线性基求交“，实际上使用正交线性基相关技巧可以很好地解决这一类问题。

形式化地：给定序列 $S_1, S_2 ... S_k$，判断值 x 是否同时作为 $S_1 \sim S_k$ 的某个子序列的异或和出现。

这相当于判断所有 $v(S_1) \cdot v(S_2) \cdot v(S_3) ... v(S_k)$，$x$ 项系数是否为零。但 $v(S_i)$ 全部逐点相乘对应的含义我们并不熟悉，我们熟悉的是将所有 $v(S_i)$ 做异或卷积得到的 $\prod_{i=1}^{k} v(S_i)$，这无非是将所有 S_i 拼成一个大序列后，大序列的点值序列。

根据 FWT 的性质，任取点值序列 $a_1 \sim a_k$，$\forall j \in [0, 2^m)$，有 $(\mathrm{FWT}(\prod_{i=1}^{k} a_i))_j = \prod_{i=1}^{k} (\hat{a}_i)_j$。考虑代入 $a_i = \hat{v}(S_i)$，那么 $(\mathrm{FWT}(\prod_{i=1}^{k} \hat{v}_i))_j = \prod_{i=1}^{k} 2^m v(S_i)_j$，这是因为对 $v(S_i)$ 做两次 FWT 后每项变为原来的 2^m 倍。

注意到 $\hat{v}_i$ 这个点值序列实际上可以找到其对应的原序列：就是 S_i 的任意一组正交线性基。

这就凑出了我们想要的 $v_1 \cdot v_2 \cdot v_3 ... v_k$ 的形式，这表明它 $v_1 \cdot v_2 \cdot v_3 ... v_k$ 的第 j 项不为零，当且仅当将所有 $\hat{v}_i$，即 S_i 的正交线性基所有元素拼接成大序列后，大序列的正交线性基（因

为外面还有一层 FWT 变换）存在子序列异或和为 j。

换句话说，将一些线性基求交，等价于将它们全部正交之后求并，将结果再正交回来。

下面一道例题展示了线性基求交的应用：

例题 4.2. [11]

给定 n 个元素是 $[0, 2^m)$ 之间非负整数的可重集 $S_1 \sim S_n$，q 次询问每次给出 x，要求找到编号 i 最小的可重集 S_i 使得 S_i 中不存在子集异或和为 x，不存在输出 $n+1$。

$n \le 10^5, |S_i| \le 64, m \le 64; q \le 10^6$。

对 $S_1 \sim S_n$ 分别求出一组正交线性基 $S'_1 \sim S'_n$，询问相当于找出最小的 i 使得 $v(S_i)$ 的第 x 项系数为 0，即最小的 i 使得 x 与 S'_i 中的某个元素不正交。这是因为 $v(S_i)$ 第 x 项为 0 等价于 $\hat{v}(S_i)$ 做 FWT 后第 x 项为 0，等价于 x 不与 $\hat{v}(S_i)$ 这个点值序列的原序列所有元素正交，等价于 x 不与 S_i 的正交线性基 S'_i 内所有元素正交。

从小到大扫描 i，维护线性基 ans，将 S'_i 内所有元素插入 ans，询问的相当于是 ans 内何时首次出现与 x 不正交的元素，只要记录 ans 每个元素被插入的时间显然即可 $O(qm)$ 回答询问。

建立需要做 $\sum |S_i|$ 次线性基插入，求 n 次正交线性基，并将 $O(nm)$ 个元素插入线性基 ans，因此时间复杂度为 $O(nm^2 + \sum |S_i| m)$。

于是该问题在 $O(nm^2 + \sum |S_i| m + qm)$ 时间内得到完美解决。

4.5 本章小结

本章从一个更 OI 的角度介绍了正交线性基并展示了其两个角度的应用：刻画点值序列以及线性基求交。其中 4.1 是一些定义与准备工作；4.2 通过讨论点值序列进行 FWT 变换后的结果，展现了正交线性基确实非常必要；4.3 介绍了构建正交线性基的算法；4.4 介绍了线性基求交。

本章内容相对第二三章较少，这是因为正交线性基难度较高，此前受到的研究与关注较少，事实上正交线性基结合上第三章提到的常见优化技巧，应该还可以解决一些更复杂、更为有趣的问题，读者阅读完本文之后可以自行思考研究。

5 带删线性基

5.1 具有缺陷的实现方法

本节中我们来考虑如何解决以下问题：

[11] 来源：候选队互测 2022 枪打出头鸟 https://uoj.ac/problem/698

例题 5.1. [12]

动态维护 (t_i, a_i) 二元组集合 S，其中 t_i 是优先级，a_i 是 m 位二进制数。初始 S 为空，需要支持 n 次操作，分为三种：

- 插入操作：向 S 内新插入一个二元组 (t_i, a_i)。
- 删除操作：删除 S 内一个已有的二元组，保证存在。
- 查询操作：给出 lim，询问如果按照任意顺序写下 S 内所有 $t_i \le lim$ 的 a_i，得到的序列的所有子序列元素异或和种类数（即构成的集合大小）mod998244353。

$n, m \le 2000$。

在 3.5 介绍用前缀线性基实现“脆弱”的动态线性基时就已提到过，即使这个问题不强制在线，用前缀线性基也必然无法解决。如果附带上每个二元组被删除的时间变为三元组，额外信息有 t_i 对应的优先级信息和“被删除的时间”两维，询问相当于希望求一个前缀矩形，即 x, y 坐标均小于等于对应上界的所有点的线性基信息，显然无论对点集如何排序都不能总用一段前缀表示这个矩形信息。

为了避免标号上的麻烦，可以按照插入操作的顺序给每个被插入的二元组一个递增的标号，插入操作视作是将一个二元组序列中 a_i 为全零二进制数的位置修改为任意值，删除操作视作是将一个二元组序列中一个在此前操作被修改为任意值的位置的 a_i 修改回全零二进制数。

解决这个问题大体上的思路仍然是以 t_i 为优先级建前缀线性基，方便起见假设 t_i 互不相同，因为实际上对于 t_i 相同的值任意给这些 t_i 定序以区分，这样求出的结果都是正确的。

难点无疑在于删除操作，考虑通过维护额外信息来支持删除操作。

额外对二元组序列的每个位置 i 维护 $resp_i$ 为 n 位二进制数，表示 a_i 如果要写作一些线性基中的元素的异或和，应包含原二元组序列的哪些元素。以及对线性基的每一位维护 $wresp_i$ 也是 n 位二进制数，表示这个值要写成一些线性基中元素的异或和，应包含原二元组序列哪些元素。

有必要做概念上的明确：线性基数组上每一位的值事实上是不直接对应原序列的值，而是原序列一些元素的异或和。假设我们按优先级 t_i 从小到大顺序插入序列中所有元素，会有一些元素插入成功，最后填上了线性基数组的某一位，接下来我们称所有这样插入成功的元素的原始值 a_i 构成的集合为一组 **标准基**。那么原序列所有子集的异或和都可以被唯一表示为标准基的一个子集的异或和，$resp_i, wresp_i$ 就分别状压了 a_i 和 w_i 进行这样分解后的子集结果，w_i 按惯例指线性基中控制第 i 位的值。

在本节的问题中，只要顺利维护了每个 a_i 是否在标准基内，显然就可以回答所有询问，因此可以使用以下带有缺陷的实现方法。

[12] 来源：原创题

要删除 (t_i, a_i)，如果 a_i 不在标准基内很简单，直接删除即可。假设 a_i 在标准基内，本来在标准基内的元素不会变得不在标准基内，只可能有唯一一个 t_j 最小但仍大于 t_i 的 j，如果其的标准基表示中包含 i，其会被加入标准基。

这两点通过考虑，(t_i, a_i) 在标准基内当且仅当 a_i 不能写成一些 $t_j < t_i$ 的 a_j 的异或和即可看出。

而找到这个 j，只需要枚举所有不在标准基内的元素 a_j，找到 t_j 最小的 $resp_j$ 含有 i 的 j 即可。

如果顺利找到了，需要在标准基中用 a_j 换掉 a_i。这会对所有 $resp_k, wresp_k$ 产生一些影响：记 $msk = 2^j \oplus resp_j$，对于所有 $resp_k$ 和 $wresp_k$，如果其本来含有 i 这一位，需要异或上 msk。而 w 数组在本节讨论范围内可以认为没有发生变化。

剩下的情况是任何 $resp_j$ 都不包含 i 。这说明除了 i 以外，任何 a_j 在标准基内的表示都不需要 i，所以可以直接将 a_i 从标准基中删去，从而也不需要修改 $resp$ 序列。但 $wresp$ 序列和 w 序列仍然会受到影响。

考虑找到 $wresp$ 最低的一位 $wres_q$ 包含 i，将高于 q 的所有包含 i 的 $wresp$ 位的值异或上 $wres_q$，w 也做相应处理。这样就保证了所有 w 序列包含的值都在删去 a_i 后仍然可以被表示为 a 序列的子集异或和，并且注意到因为 q 是最低的 $wresp$ 包含 i 的位，所以更高的 w 序列的位的值不会受影响（在先前的处理中只异或上了 w_q，而 w_q 的最高位是 q，不会违背线性基的定义），最后将 $w_q, wresp_q$ 置零即可。

接下来考虑插入操作，因为标准基必须是按优先级 t_i 从小到大顺序插入时插入成功的 a_i 构成的集合，所以注意如果直接向线性基插入插入 a_i 失败了，不代表 a_i 不在标准基中：把 a_i 写成标准基子集异或和的形式，如果这个子集中 t_j 最大的项的 $t_j > t_i$，那么应该把 a_j 从标准基中删去，将 a_i 加入标准基。这样做的正确性分析和删除时的正确性分析一致，执行这样的操作对 $resp, wresp$ 会产生与删除部分类似的影响，处理方法也是一致的。把 a_i 写成标准基子集异或和的形式这一步通过 $wresp$ 数组即可完成。

审视插入和删除过程，可以发现执行了 $O(m)$ 次长度为 m 的二进制数的异或操作，以及 $O(n+m)$ 次长度为 n 的二进制数的异或操作，其余部分代价不超过 $O(n+m)$。注意由于该算法维护的二进制数长度是 n 或 m 的，而 n, m 显著比计算机字长 w 大，所以应当认为对这些二进制数进行异或运算的时间复杂度是 $O((n+m)/w)$ 而非 $O(1)$，因此插入和删除的时间复杂度均为 $O((n+m)^2/w)$。

于是在 $O((n+m)^2/w)$ 时间内解决了这一问题。

这种带删线性基的最大缺陷在于单次插入和删除的时间开销带有关于序列长度 n 的因子，且这实际上是完全无法避免的，在 n, m 不同阶时这会导致这种带删线性基几乎完全无法使用。

但就算抛开这个缺陷不谈，这种实现方法仍然还有巨大的缺陷：这样实现的带删线性基的“前缀”性是不完全的。如果将询问的内容改为“所有子序列异或和的最大值”，这一实现方法实际上就失效了：维护的 w 事实上仍然不带有“前缀”的性质，仍然是一种全局

信息。这种实现方法只保证了能够维护顺利维护所有使得线性基扩张的位置，只有在这一信息上才具有“前缀”性。

虽然有这两个缺陷，这种带删线性基也已经可以解决一些以往无法解决的问题，除本节例题以外最简单的例子就是即使目标是维护所有子序列异或和最大值，只要不管 *lim* 的限制即保持 *lim* 足够小，这个算法就完全是正确的，并且是在线算法，不是 3.5 中描述的“虚假”的前缀线性基可以媲美的。

除此之外，用它还可以解决一些更为复杂的问题，将在 5.2 中介绍。

5.2 带删前缀线性基结合数据结构

回顾 3.2 中描述的问题：

例题 5.2.

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a。支持 q 次操作分为两种：

- 修改操作：将某个 a_x 修改为 y，保证 $y \in [0, 2^m)$。
- 查询操作：给出区间 $[l, r]$，查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \le 5 \times 10^4, m \le 30$。

3.7 中描述的问题及其解法，可以认为是解决了本题保证每次修改前 a_x 均为 0 的特殊情形，现在延续 3.7 中的思路并使用 5.1 中介绍的（即使是带有缺陷的）带删线性基，可以给出该问题时间复杂度为 $O(nm + qm(\log n + m))$ 的解法。

既然要使用带删线性基，迫在眉睫的问题就是如何解决其时间开销带有其维护的序列长度 n 作为因子的问题。回顾 3.7 中的做法，实际上是对线段树上每个管辖区间 $[l, r]$ 的节点，维护两个前缀线性基，分别维护二元组序列 $\forall l \le i \le r, (i, a_i)$ 与 $\forall l \le i \le r, (-i, a_i)$。现在我们指出最关键的一点：线段树是分治结构，其每个节点的信息可以由其子节点信息合并而来，反映到这个问题上，就是父节点 u 上的前缀线性基可以认为维护的二元组序列并非所有 $\forall l \le i \le r$ 的 (i, a_i)，而是其两个子节点 $ls(u), rs(u)$ 的标准基序列，而后者的长度受到线性基大小上界的控制，至多只有 $2m$。

于是现在终于得以将 3.7 的算法拓展到可以将某个非零元素修改为其他任意值的情况。对线段树上所有节点维护正反两个前缀带删线性基，对于非叶节点，认为维护的二元组序列是其两个子节点的标准基序列；而对于叶节点，这个线性基只包含一个元素当然可以直接得到。

所有结构的建立是简单的，叶子节点可以直接建立，非叶子节点将左右儿子的标准基的各个元素依次插入即可。

修改时，叶节点的标准基的变化可以直接得到，将这些变化记录为若干次向其两个方向的标准基内插入或删除二元组的事件。如果对于某个子节点已经得知了所有这些事件，在其父节点的带删前缀线性基看来，相当于父节点的带删前缀线性基要维护的二元组序列发生了一些单点修改，执行这些单点修改又会导致其标准基发生若干次插入和删除事件，不断向上传递影响即可。事实上对一个序列的值做单点修改，对这个序列的标准基产生的影响当然只是 $O(1)$ 次插入和删除。所以把每次记录下来的插入和删除事件做一些抵消（要插入和删除相同的值可以都消去）就可以保证插入和删除事件每层都只进行 $O(1)$ 次。

查询时，像 3.7 中的做法一样一样找到询问区间 $[l,r]$ 被首次切开的位置 mid，相当于要查询 $[L,mid],[mid+1,R]$ 两段区间的线性基信息。比如 $[mid+1,R]$ 的部分，其实直接取出 $[mid+1,r]$ 这一线段树节点维护的标准基中，所有编号 $\le R$ 的元素，插入一个新的线性基即可。这里使用 5.1 中介绍的“前缀”性不完全的带删线性基确实就足以解决问题，且就算换成“前缀”性完全的带删线性基也必然不会优化复杂度，因为最后无论如何都要执行一次 $O(m^2)$ 的线性基合并，而取出标准基的不超过 m 个元素重新插入到一个线性基中的时间开销也只有 $O(m^2)$。

于是修改最终被拆成 $O(m)$ 次对带删线性基进行插入与删除操作，且保证了用到的带删线性基维护的序列长度也都是 $O(m)$ 的，因此插入的时间复杂度为 $O(m\log n)$。查询时求 LCA 的代价可以忽略不计，时间开销是从左右两边各找出至多 m 个标准基元素插入到用于求答案的线性基中，时间复杂度为 $O(m^2)$。

但不幸的是这样实现会导致建立的时间复杂度以及空间复杂度都较高，为此还可以加上一个优化：底层分块，将线段树上所有管辖的区间 $[l,r]$ 满足 $r-l+1\le m$ 的节点视为叶节点，在线段树上去掉其所有后代节点。修改直接 $O(m^2)$ 重建唯一的一个这样的伪叶节点的标准基信息然后再不断传递影响，查询时如果询问区间不被某个伪叶节点管辖的区间完全包含那么就完全不受影响，否则说明询问区间长度不超过 m，直接暴力建立询问区间的线性基即可 $O(m^2)$ 回答询问。

而底层分块降低了建立的时间复杂度以及该算法的空间复杂度，首先建立所有伪叶节点的标准基信息当然时间复杂度为 $O(nm)$，对于剩下的非伪叶节点，其维护的二元组序列长度为 $O(m)$，因此建立时间复杂度为 $O(m^2)$，而线段树上维护区间长度超过 m 的节点只有 $O(\frac{n}{m})$ 个，因此建立的时间复杂度为 $O(nm)$。而该算法只维护了 $O(\frac{n}{m})$ 个带删前缀线性基，且每个带删前缀线性基维护的序列长度为 $O(m)$，因此空间复杂度是 $O(n)$ 的。

综上所述，该问题可以在 $O(nm+qm(\log n+m))$ 时间，$O(n)$ 空间内在线解决。相比 3.2 中描述的传统算法在时间复杂度和空间复杂度上都远更优秀，唯二的缺点在于思路与实现较为复杂，以及常数较劣。

5.3 具有完全前缀性的实现方法

接下来我们来介绍具有完全前缀性的实现方法。这一实现方法与带有缺陷的实现方法具有一定相似性，仍然是通过维护 $resp,wresp$ 这些额外信息来试图实现删除操作，但区别

在于其对线性基的形态做了更严格的约束，首先要求其具有前缀性，在具有前缀性的基础上，还规定了更严格的限制，以保证按任意顺序插入二元组得到的线性基形态（主要指的是 $w, wresp$ ）完全一致，而 5.1 中带有缺陷的实现方法对 $w, wresp$ 的限制实际上是非常宽松的。

另外，一般的前缀线性基实际上如果按照不同顺序插入二元组，得到的线性基形态也可能不相同。如果希望验证这一点，读者可以自行编写代码按照随机顺序将二元组插入前缀线性基，得到的线性基事实上几乎总是不相同。

然后，利用这个重要的“完全一致”性，可以认为在删除操作中，被删除的二元组实际上是最后一个被插入的二元组，从而认为本次删除操作其实是在撤销最后一次插入操作。但与真正的撤销最后一次插入操作相比，常规的“在插入时记录一些信息作为线索，以帮助此后的撤销”这样的方法完全不适用，因此在删除过程中只能通过维护的额外信息 $resp, wresp$ 作为线索来尝试还原这最后一次插入中究竟进行了哪些操作。

方便起见，本节只考虑以下问题：

例题 5.3. [13]

维护长度为 n，由 n 位二进制数组成的序列 a，支持 q 次操作分为两种：

- 修改操作：给出位置 p 以及 n 位二进制数 val，将 a_p 修改为 val。
- 查询操作：给出位置 p，查询 $a[1, p]$ 所有子序列中，异或和最大的子序列。

$n, q \leq 2000$。

既然我们希望按任意顺序插入二元组得到的线性基形态完全一致，总得先明确这样的形态是什么。这里我们就希望得到的线性基形态是按照编号 $i = 1 \sim n$ 的顺序按照 2.3 中描述的最朴素的方式将 a_i 插入线性基，从而得到的 w 数组以及对应的 $wresp$ 方案。

但这种描述的性质太差了，实际上可以注意到这样做与下面更复杂但性质更好的描述等价：

仍然按照 $i = 1 \sim n$ 的顺序考虑，于是在考虑 a_i 前，所有 $\forall 1 \leq j < i$ 的 a_j 是如何插入线性基的已经确定。对于 a_i，考虑将其异或上 $a[1, i)$ 的一个子序列使得得到的值 val 的**最高位**尽可能小，当然这样的选择方案可能有很多种，val 的值也不确定。如果 val 可以为 0，那么 a_i 当然不在标准基中，在 w 中也不存在其对应的元素，其对于 w 没有影响。否则设 val 的最高位为 p，我们对选取 $a[1, i)$ 子序列的方式做出限制：首先必须选在标准基内的元素，其次选出的标准基内的元素必须作为某个 $p < k < n$ 的 $wresp_k$ 的最高位出现。可以发现，这样就限定了选出的 $a[1, i)$ 子序列是唯一的，w_p 必须是 a_i 异或上这样的唯一子序列的异或和，$wresp_p$ 同样描述对应的方案。因为这里的描述涉及 $wresp_k$ 的最高位，下面记 $wresp_k$ 的最

[13] 来源：原创题

高位为 idx_k。特别地，如果 $w_k = 0$，认为 $idx_k = +\infty$。可以发现 idx_k 其实就表明了 w_k 首次被填上值时，对应的 i 是多少。

仍然将修改操作视作一次插入操作（保证 a_p 修改前全 0）和一次删除操作（认为修改为的值 val 全零），既然希望将删除视作撤销最后一次插入，先考虑插入操作如何进行。

现在的线性基形态是限制更严格的前缀线性基，因此插入操作大体上与前缀线性基的插入是类似的。维护 $cur, cresp, q$ 分别表示当前要插入的值，这个值真正作为原序列子序列异或和时的表示方法，以及当前要插入的值对应的下标。初始 cur 为要插入的二进制数，$cresp = 2^p$，$q = p$。降序考虑 i 从 $n-1$ 到 0：

- 若 cur 的第 i 位为 0，那么注意 $idx(i, n)$ 构成的集合中，只有 p 被加入了集合，以及 q 被从集合中删除（如果 $p = q$，即无事发生）。那么如果 $wresp_i$ 是包含 q 的，应该用 $cresp$ 所对应的包含 p 和 q，且第 $i \sim n-1$ 位均为零的的子序列异或和表达式从 $wresp_i$ 中消去 q，w_i 也相应要异或上 cur，来保持更严格的限制。如果 $wresp_i$ 包含 q，隐含了 $q < idx_i$，所以异或之后 idx_i 当然不会发生变化。

  注意这一条体现了限制比前缀线性基更严格之处。

- 接下来是 cur 的第 i 位为 1 的情况。和前缀线性基一样，这里要根据 idx_i 和 q 的大小关系来考虑要不要交换 cur, w_i 以及 $cresp, wresp_i$，还有 q, idx_i。无论是否交换，将 cur 异或上 w_i，$cresp$ 异或上 $wresp_i$。这里的疑惑可能在于如果 $q > idx_i$，那么不需要交换，但需不需要和 cur 的第 i 位为 0 中的情况一样检查是否 $wresp_i$ 是否包含 q。但再回顾一下定义：idx_i 是 $wresp_i$ 的最高位，即可发现这当然不需要检查。

  如果这一步结束之后发现 $q = +\infty$，需要立即退出。

在退出之后，如果 $q = +\infty$ 说明只有 p 被加入了标准基，标准基中没有发生任何删除。于是不需要检查所有不在标准基内的元素关于标准基内元素的表法是否有一些变得不合法。否则对于 $1 \le q \le n$ 的情况，说明 p 被加入了标准基，q 本来在标准基中但现在被删除（$p = q$ 会对应一个插入了 p 但它不在标准基中的无意义情况，可能会比较奇怪，判掉就好了），通过 $cresp$ 就得到了 $resp_q$ 这一将 q 表示为一些标准基内元素的表法，同时还需要检查所有不在标准基内的元素，如果其 $resp_i$ 包含 q，需要异或上 $cresp$ 来消去 q。

接下来考虑删除操作，因为希望将其视作撤销最后一次插入，我们需要先还原在最后一次插入结束后，$cur, cresp, q$ 三者的具体值。首先需要区分的两种情况是插入后标准基中发生了删除的情况，与没有发生删除的情况。这一步与 5.1 中找出删除后哪个元素被加入标准基的方式是完全一致的：枚举 $\forall p < i \le n$，若 a_i 不在标准基内且 $resp_i$ 包含 p，取其中最小的 i 就是最后一次插入结束后的 q，即唯一一个从标准基内删除的元素，如果不存在说明插入结束后 $q = +\infty$。如果 $1 \le q \le n$，顺便也得知了 $cur = 0$，$cresp$ 是 $resp_q$ 翻转第 q 位后的结果；$q = +\infty$ 时由于循环在某个 i 就做了立刻退出，我们还希望还原这个 i。

在这里需要注意，对于 $1 \le q \le n$ 的情况需要做没有本质困难但必不可少的对 $resp_i$ 修改的撤销。正着做的过程是给所有 $resp_i$ 含有 q 的 $resp_i$ 异或上 $cresp$，同时注意到 $cresp$ 必

然含有 p，且 $resp_i$ 在异或前必然不含 p。于是倒着做的过程就是给所有 $resp_i$ 含有 p 的 $resp_i$ 异或上 $cresp$。

在这里需要介绍一个重要性质：在最后一轮插入 a_p 时，cur 任何时刻都一定包含 p 这一位，而在插入 a_p 前所有 $resp_i, wresp_i$ 都一定不包含 p 这一位，所以插入后所有 $resp_i, wresp_i$ 是否包含 p 这一位实际上是非常重要的标识符，其起到了提示插入过程，作为线索的作用。观察插入过程，每当 $cresp, wresp_i$ 发生交换时，$wresp_i$ 会从本来不含 p 这一位变得含有 p 这一位。另一种让 $wresp_i$ 含有 p 这一位的方式是在插入到第 i 位时 cur 的第 i 位为 0，此时如果 $wresp_i$ 包含当时的 q，$wresp_i$ 会被异或上 $cresp$ 从而含有 p 这一位。$wresp_i$ 只能通过这两种方式含有 p 这一位。

那么在 $q = +\infty$ 时还原退出的 i 实际上就非常简单了：退出的 i 一定是最后一次 $cresp, wresp_i$ 发生交换的位置，于是 $wresp_i$ 一定含有第 p 位，而 $\forall 0 \le j < i$ 的 $wresp_j$ 当然不含有第 p 位。得到退出的 i 之后，在这里做一些处理之后，可以认为 $1 \le q \le n$ 和 $q = +\infty$ 两种情况接下来没有区别。

接下来考虑如何在每一步过程中倒推，设现在已知 $cur, cresp, q$ 是插入过程中进行完第 i 轮之后的值，考虑如何通过 $w_i, wresp_i$ 还原出 $cur', cresp', q'$ 是插入过程中进行完第 i 轮之前的值，并将 $w_i, wresp_i$ 也还原到进行完第 i 轮之前。一个非常平凡的情况是 $w_i = 0$，那么在插入前一定也 $w_i = 0$，且插入过程对 $w_i, wresp_i$ 没有发生任何影响，于是可以直接跳过这一位。

下面所有带引号的表示执行插入操作第 i 轮前的值，不带引号的表示执行完插入操作第 i 轮后的值。

我们先假设已经通过某种手段，判断出了进行插入的第 i 轮前 cur' 的第 i 位是 0 还是 1，并分别考虑在这两种情况下如何还原。

- cur' 的第 i 位是 0：那么需要判断出插入的第 i 轮前，是否 $wresp'_i$ 包含 q'，从而 $wresp'_i$ 和 w'_i 分别异或上了 $cresp', cur'$。但其实注意到如果异或了，那么 $wresp_i$ 必然含有 p 这一位，否则必然不含，于是这部分的还原可以完成。

- cur' 的第 i 位是 1：那么需要判断插入的第 i 轮中，是否发生了交换。还是从标识符的角度考虑，如果交换了必然 $wresp_i$ 含有 p，否则必然不含，于是这部分的还原也可以完成。

现在最终的困难在于如何确定插入的第 i 轮前 cur' 的第 i 位的值。我们依次检查以下几点：

- 如果 $cresp$ 包含 idx_i 这一位，可以发现必然 cur' 的第 i 位为 1。但对于 $cresp$ 不包含 idx_i 的情况，有可能是一般的 cur' 的第 i 位为 0 的情况；也可能是 cur' 的第 i 位为 1，但 $q' < idx'_i$ 于是发生了交换，而本来的 $wresp'_i$ 就是包含 q' 的，交换后异或反而消除了。

- 不然，如果 $wresp_i$ 不包含 p，可以发现 cur' 的第 i 位必然是 0。因为现在假设 cur' 的第 i 位为 1，必然是发生了交换的分支，而这种情况必然 $wresp_i$ 包含 p。

- 如果还是无法区分，假设 cur' 的第 i 位为 0，那么必然使用 $cresp'$ 执行了对于 q' 的消去，此时隐含了 $q' < idx'_i$，于是 $cresp = cresp'$ 在 $idx_i = idx'_i$ 及之后的位上均必然为 0。而假设 cur' 的第 i 位是 1，那么必然发生了 q' 与 idx'_i 之间，$cresp'$ 与 $wresp'_i$ 之间的交换，于是 $cresp = wresp'_i$ 在 idx'_i 这一位上必然为 1，而 $idx'_i > idx_i = q'$，因此 $cresp$ 一定在某个 idx_i 之后的位上为 1。只需检查 $cresp$ 的最高位是否不低于 idx_i 即可分辨。

现在所有需要的信息均得到了还原，因而删除操作可以被顺利地视作撤销最后一次插入操作并执行。无论是插入操作和删除操作，时间瓶颈均为执行 $O(n)$ 次长度为 n 的二进制数的异或操作，因而单点修改操作可在 $O(n^2/w)$ 时间内完成。

既然顺利维护了比一般前缀线性基限制更强的线性基形态，查询操作也可以在 $O(n^2/w)$ 时间内轻易解决。事实上将查询的内容换成其他线性基可维护的经典信息该算法也完全可以胜任。

由于建立还需要 $O(n^3/w)$ 时间，综上所述，该例题在 $O((n+q)n^2/w)$ 时间内得以在线解决。

5.4 本章小结

本章聚焦于在线性基相关问题中极其常见的带删线性基问题，重点讨论了维护的集合大小与需要维护的二进制数长度同阶的情形。事实上带删线性基在第三章就已被讨论过：

- 3.2 的例题在询问均查询一段前缀时便可视为和 5.3 中例题类似的带删线性基问题，但当时讨论的设定是需要维护的集合大小远大于需要维护的二进制数长度，因而线性基合并的时间开销虽然较大但仍可接受，因此当时使用的方法其实是避免执行删除，利用线段树的结构全部转化为线性基合并。

- 3.5 中的例题三也是带删线性基问题的形式，但该例题并没有强制占用掉线性基的“前缀”性，即查询时总查询整个集合，且也并不强制在线，因此可以利用前缀线性基的技巧以删除时间为优先级避免真的执行删除操作。

所以此前的方法均为试图寻求转化，避免直面删除操作。但事实上这些转化的方法大多非常脆弱：对于全部转化为合并的方法，就算抛开转化为合并本身乘上的 $\log n$ 倍的操作代价不谈，在维护的二进制数位长较大时急剧增加的合并代价也会使这种方法完全无法使用（合并需要 $O(m^2)$ 次长度为 m 的二进制数运算，而常规插入只需要 $O(m)$ 次长度为 m 二进制数运算）；而对于使用前缀线性基技巧，以删除时间为优先级建立前缀线性基的方法，简单加上强制在线即可让这种方法完全束手无策。

因此本章在 5.1 和 5.3 介绍了两种功能强大的带删除线性基。因为“前缀”性是线性基所具有的非常重要的性质，还着重考虑了“前缀”性在两种带删除线性基上能否得以保持。其中 5.1 一节介绍的带删线性基原理、实现较为简单，但具有缺陷，只具有部分“前缀”性。5.3 一节介绍的带删线性基基于 5.1 中介绍的带有缺陷的版本并加以改进，使其具有完全“前缀”性，但原理、实现更为复杂。

但需要注意的是，我认为带删线性基单次操作的时间开销无法避免地与其维护的元素个数有关，这导致其在传统的需要维护的集合大小远大于要维护的二进制数长度的情形下并不优秀，因此类似的方法此前也并未引起广泛关注。

具体而言，我尝试在互联网上搜索带删线性基，搜索到的关于带删线性基的内容只有 UOJ453 这道例题及其衍生博客，其中使用的带删线性基与 5.1 一节中介绍的类似，但其不具有任何“前缀”性。我在这种带删线性基基础上加以改进，先后发明了 5.1,5.3 中介绍的两种具有部分“前缀”性和完全“前缀”性的带删线性基，应当在一定程度上填补了这部分研究的空白。

本章剩下的 5.2 一节介绍了如何使用 5.1 中介绍的具有部分“前缀”性的带删线性基配合线段树优化 3.2 中提到的经典问题，得到的新解法相比传统做法少一个 log 因子，该做法（据我所知的范围内）同样首次由我提出。

本章正文只有三节，相比二三四章均更少，且均为本文原创内容，难以在网络上搜索到相关资料。这表明了带删线性基这一方向受到的研究仍然较少，希望读者在这一方向多多思考，创造出更多成果！

6 总结

本文依次介绍了线性基基本的构建方法与基础应用、线性基上的常见优化技巧、正交线性基、带删线性基。前三部分主要对信息学竞赛中已有的理论做规范化的梳理整合，最后一部分在带删线性基这一此前较少受到研究的领域提出了一些全新的理论，对一些经典问题做出了优化。

希望本文能起到抛砖引玉的作用，给与读者在这一方向一些启发，也希望未来能出现更多线性基的有趣题目与拓展。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师的教导和同学们的帮助。

感谢章绍嘉、汪苏轶同学与我进行交流讨论，并为本文审稿。

感谢其它所有本文所参考过的资料的提供者。
感谢各位百忙之中抽出宝贵的时间来阅读本文。

参考文献

[1] 苏焜, 浅谈「正交线性基」https://www.luogu.com/article/qb29lo53

[2] 吴与伦, 动态线性基学习笔记 https://mem.ac/oi/algorithm/dynamic-linear-basis/