Menci's Blog

幻梦终醒，不悔华年

线性基学习笔记

2017-02-25 | OI

线性基是竞赛中常用来解决子集异或一类题目的算法。

定义

异或和

设 S 为一无符号整数集（下文中除非特殊说明，**集合**均指「无符号整数集」），定义其**异或和** $\text{xor-sum}(S) = S_1 \text{ xor } S_2 \text{ xor } \ldots \text{ xor } S_{|S|}$。

张成

设 $T \subseteq S$，所有这样的子集 T 的异或和组成的集合称为集合 S 的**张成**，记作 $\text{span}(S)$。即，在 S 中选出任意多个数，其异或和的所有可能的结果组成的集合。

线性相关

对于一个集合 S，如果存在一个元素 S_j，使得，S 在去除这个元素后得到的集合 S' 的张成 $\text{span}(S')$ 中包含 S_j，即，$S_j \in \text{span}(S')$，则称集合 S **线性相关**。

更形象地，可以表示为，存在一个元素 S_j，可以用其它若干个元素异或起来得到。

相对的，如果不存在这样的元素 S_j，则称集合 S **线性无关**。

一个显然的结论是，对于这个线性相关的集合 S，去除这个元素后，集合的张成不变。

线性基

我们称集合 B 是集合 S 的线性基，当且仅当：

1. $S \subseteq \text{span}(B)$，即 S 是 B 的张成的子集；
2. B 是线性无关的。

集合 B 中元素的个数，称为线性基的**长度**。

线性基有以下基本性质：

1. B 是极小的满足线性基性质的集合，它的任何真子集都不可能是线性基；
2. S 中的任意元素都可以**唯一**表示为 B 中若干个元素异或起来的结果。

构造与性质

这里讲解一种线性基的构造方法，及其特殊性质，之后，我们将讨论这种构造方法的正确性。下文中「线性基」均指这种方法构造出的线性基。

设集合 S 中最大的数在二进制意义下有 L 位，我们使用一个 $[0 \dots L]$ 的数组 a 来储存线性基。

这种线性基的构造方法保证了一个特殊性质，对于每一个 i，a_i 有以下两种可能：

1. $a_i = 0$，并且
   - **只有**满足 $j > i$ 的 a_j（即，位于 a_i 后面的所有 a_j）的第 i 个二进制位**可能**为 1；
2. $a_i \neq 0$，并且
   - 整个 a 数组中**只有** a_i 的第 i 个二进制位为 1；
   - a_i 更高的二进制位（$> i$ 的二进制位）**一定**为 0；
   - a_i 更低的二进制位（$< i$ 的二进制位）**可能**为 1；

我们称第 i 位**存在于**线性基中，当且仅当 $a_i \neq 0$。

举个例子，我们的构造方法可能得到一个这样的数组 a（右侧为最低二进制位），其中被标为蓝色的元素为上述的第 2 种情况，绿色的元素为上述的第 1 种情况。

$$
\begin{aligned}
a_0 &\quad 0\,0\,0\,0\,1 \\
a_1 &\quad 0\,0\,0\,0\,0 \\
a_2 &\quad 0\,0\,1\,1\,0 \\
a_3 &\quad 0\,0\,0\,0\,0 \\
a_4 &\quad 1\,1\,0\,1\,0
\end{aligned}
$$

举两个反例：

$$
\begin{aligned}
a_0 &\quad 1\,0\,0\,1 \\
a_1 &\quad 0\,0\,1\,0 \\
a_2 &\quad 0\,1\,0\,0 \\
a_3 &\quad 1\,0\,0\,0
\end{aligned}
\qquad\qquad
\begin{aligned}
a_0 &\quad 0\,1\,0\,1 \\
a_1 &\quad 0\,0\,1\,0 \\
a_2 &\quad 0\,0\,0\,0 \\
a_3 &\quad 1\,0\,0\,0
\end{aligned}
$$

首先，线性基是动态构造的，我们只需要从空的 a 开始，每次考虑在一个已存在的线性基中插入一个数 t 即可。

从 t 最高位上的 1 开始考虑，设这是第 j 位，如果这一位已经存在于线性基中，则我们需要将 t 中的这一位消掉（将 t 异或上 a_j），才可以继续插入（因为只有 a_j 的第 j 位可以为 1）。如果这一位不存在于线性基中，则可以将 t 插入到 a_j 的位置上，但插入时需要保证：

1. t 中比第 j 位更**高**的**已经存在于线性基中的**二进制位上必须为 0，而这时候 t 的最高位上的 1 一定是第 j 位（更高位上即使原本有，也被消掉了），所以**无需考虑**；
2. t 中比第 j 位更**低**的**已经存在于线性基中的**二进制位上必须为 0，对于这一点，我们可以枚举 t 中为 1 的这些二进制位 k（$k \in [0, j)$）对应的元素 a_k，并用类似上面的方式将 t 异或上 a_k，以消掉这些位上的 1；
3. a 中必须只有 a_j 的第 j 位为 1：
   - a 中在 a_j 前面的元素的第 j 位必须为 0，这一点必定已经满足，假设存在一个 $k \in [0, j)$ 满足 a_k 的第 j 位为 1，则 a_k 在插入时就应该被插入到 a_j 的位置上，所以**无需考虑**；
   - a 中在 a_j 后面的元素的第 j 位必须为 0，对于这一点，我们可以枚举 a_j 后面的元素 a_k（$k \in (j, L]$），将每个第 j 位为 1 的 a_k 异或上 t。

流程

- 逆序枚举 t 所有为 1 的二进制位 $j = L \to 0$，对于每个 j
  - 如果 $a_j \neq 0$，则令 $t = t \operatorname{xor} a_j$
  - 如果 $a_j = 0$，则
    - 枚举 $k \in [0, j)$，如果 t 的第 k 位为 1，则令 $t = t \operatorname{xor} a_k$
    - 枚举 $k \in (j, L]$，如果 a_k 的第 j 位为 1，则令 $a_k = a_k \operatorname{xor} t$
    - 令 $a_j = t$，结束插入过程

代码

```
const int MAXL = 60;

struct LinearBasis
{
    long long a[MAXL + 1];

    LinearBasis()
    {
        std::fill(a, a + MAXL + 1, 0);
    }

    void insert(long long t)
    {
        // 逆序枚举二进制位
        for (int j = MAXL; j >= 0; j--)
        {
            // 如果 t 的第 j 位为 0，则跳过
```

```
            if (!(t & (1ll << j))) continue;

            // 如果 a[j] != 0，则用 a[j] 消去 t 的第 j 位上的 1
            if (a[j]) t ^= a[j];
            else
            {
                // 找到可以插入 a[j] 的位置

                // 用 a[0...j - 1] 消去 t 的第 [0, j) 位上的 1
                // 如果某一个 a[k] = 0 也无须担心，因为这时候第 k 位不存在于线性基中，不需要保
                for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];

                // 用 t 消去 a[j + 1...L] 的第 j 位上的 1
                for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;

                // 插入到 a[j] 的位置上
                a[j] = t;

                // 不要忘记，结束插入过程
                return;
            }

            // 此时 t 的第 j 位为 0，继续寻找其最高位上的 1
        }

        // 如果没有插入到任何一个位置上，则表明 t 可以由 a 中若干个元素的异或和表示出，即 t 在
    }
};
```

证明

首先，根据归纳法，可以得出其特殊性质是满足的。

我们枚举 S 中的所有元素作为 t，分别插入，并将数组 a 转化为一个集合 B（只需要去除重复的为 0 的元素即可，显然不为 0 的元素不会重复），此时 B 就是 S 的线性基：

首先，在插入的过程中，我们每次找到 t 最高位上的 1，然后把它消去，如果最终全部被消去，则表示要插入的 t 已经可以由 a 中一些元素的异或和表示出，此时不需要插入。这样保证了 B 是线性无关的。对于被实际插入到 a 中的元素，它们在实际插入之前做了一些变换，这些变换都是使它异或上 a 中已存在的元素，或者使 a 中已存在的元素异或上它 —— 这些变换都是可逆的，所以用 a 中一些元素的异或和可以表示出这些（实际被插入的）元素。

同时，显然该算法的时间复杂度为 $O(\log t)$ 单次插入，空间复杂度同样为 $O(\log t)$。

合并

两个集合的线性基可以在 $O(\log^2 t)$ 的时间内进行合并，合并后得到的线性基为两个集合的并的线性基。

合并只需要将一个线性基中的所有元素插入到另一个线性基中即可。

```
void mergeFrom(const LinearBasis &other) {
    for (int i = 0; i <= MAXL; i++) insert(other.a[i]);
}

static LinearBasis merge(const LinearBasis &a, const LinearBasis &b) {
    LinearBasis res = a;
    for (int i = 0; i <= MAXL; i++) res.insert(b.a[i]);
    return res;
}
```

应用

线性基最基本的应用，就是**子集最大异或和**问题：

给一个集合 S，求它的一个子集 $T \subseteq S$，使得 $\text{xor-sum}(T)$ 最大，求出这个最大值。

首先，求出 S 的线性基 B，问题转化为选择 B 的一个子集。从高到低考虑在线性基中的二进制位 j，如果第 j 位存在于线性基中，则考虑到线性基中只有惟一一个元素的第 j 位为 1，所以之前加入到 T 中的元素的异或和的第 j 位一定为 0，即，将这个元素加入到 T 中一定会使答案更大 —— 所以，求出线性基中所有元素的异或和，即为答案。

```
long long queryMax()
{
    long long res = 0;
    for (int i = 0; i <= MAXL; i++) res ^= a[i];
    return res;
}
```

- 子集 k 大异或和：HDU 3949
- 最大路径异或和：WC2011 Xor
- 求子集异或值排名：BZOJ 2844
- 树上两点间子集最大异或和：SCOI2016 幸运数字

模板

一

不显式构造出集合 B，支持动态插入。

```
struct LinearBasis
{
    long long a[MAXL + 1];

    LinearBasis()
    {
        std::fill(a, a + MAXL + 1, 0);
    }

    LinearBasis(long long *x, int n)
    {
        build(x, n);
    }

    void insert(long long t)
    {
        for (int j = MAXL; j >= 0; j--)
        {
            if (!t) return;
            if (!(t & (1ll << j))) continue;

            if (a[j]) t ^= a[j];
            else
            {
                for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
                for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;
                a[j] = t;
                break;
            }
        }
    }

    // 数组 x 表示集合 S，下标范围 [1...n]
    void build(long long *x, int n)
    {
        std::fill(a, a + MAXL + 1, 0);

        for (int i = 1; i <= n; i++)
        {
            insert(x[i]);
        }
    }

    long long queryMax()
    {
        long long res = 0;
        for (int i = 0; i <= MAXL; i++) res ^= a[i];
        return res;
    }

    void mergeFrom(const LinearBasis &other)
```

```
    {
        for (int i = 0; i <= MAXL; i++) insert(other.a[i]);
    }

    static LinearBasis merge(const LinearBasis &a, const LinearBasis &b)
    {
        LinearBasis res = a;
        for (int i = 0; i <= MAXL; i++) res.insert(b.a[i]);
        return res;
    }
};
```

二

显式构造出集合 B（代码中的 v），不支持动态插入。

```
struct LinearBasis
{
    std::vector<long long> v;
    int n; // 原有集合 S 的大小

    // 数组 x 表示集合 S，下标范围 [1...n]
    void build(long long *x, int n)
    {
        this->n = n;
        std::vector<long long> a(MAXL + 1);

        for (int i = 1; i <= n; i++)
        {
            long long t = x[i];

            for (int j = MAXL; j >= 0; j--)
            {
                if ((t & (1ll << j)) == 0) continue;

                if (a[j]) t ^= a[j];
                else
                {
                    for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
                    for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t

                    a[j] = t;
                    break;
                }
            }
        }

        v.clear();
        for (int i = 0; i <= MAXL; i++) if (a[i]) v.push_back(a[i]);
    }
```

```
    long long queryMax()
    {
        long long x = 0;
        for (size_t i = 0; i < v.size(); i++) x ^= v[i];
        return x;
    }
};
```

数学 # 学习笔记 # 算法模板 # 线性基

「SCOI2016」美味 - 贪心 + 主席树

「HDU 3949」XOR - 线性基