```cpp
#include "../bits/stdc++.h"
// binary trie に対し，全要素にある値を xor する操作を追加したもの
// verified: https://atcoder.jp/contests/dwacon5th-final-open/submissions/4650380
class BinaryTrie
{
    using T = unsigned;
    // bit 数
    static const int B = 32;
    struct Node
    {
        // 要素数
        int cnt;
        // xor する値
        T lazy;
        Node *ch[2];
        Node() : cnt(0), lazy(0)
        {
            ch[0] = nullptr, ch[1] = nullptr;
        }
    };

    Node *root;

    void eval(Node *t, int b)
    {
        if ((t->lazy >> (T)b) & (T)1)
            std::swap(t->ch[0], t->ch[1]);
        if (t->ch[0])
            t->ch[0]->lazy ^= t->lazy;
        if (t->ch[1])
            t->ch[1]->lazy ^= t->lazy;
        t->lazy = 0;
    }

    // Node t 以下に 値 val を追加
    Node *add(Node *t, T val, int b = B - 1)
    {
        if (!t)
            t = new Node;
        t->cnt += 1;
        if (b < 0)
            return t;
        eval(t, b);
        bool f = (val >> (T)b) & (T)1;
        t->ch[f] = add(t->ch[f], val, b - 1);
        return t;
    }
    // Node t 以下から値 val を削除
    Node *sub(Node *t, T val, int b = B - 1)
    {
        // 要素が無ければエラー
        assert(t);
        t->cnt -= 1;
        if (t->cnt == 0)
            return nullptr;
        if (b < 0)
            return t;
        eval(t, b);
        bool f = (val >> (T)b) & (T)1;
        t->ch[f] = sub(t->ch[f], val, b - 1);
        return t;
    }
    // Node t 以下の k-th element を取得
    T get(Node *t, int k, int b = B - 1)
    {
        assert(t);
        if (b < 0)
            return 0;
        eval(t, b);
        int m = t->ch[0] ? t->ch[0]->cnt : 0;
        return k < m ? get(t->ch[0], k, b - 1) : get(t->ch[1], k - m, b - 1) | ((T)1 << (T)b);
    }
    // Node t 以下で val と xor するときの最小値取得
    T getMinimum(Node *t, T val, int b = B - 1)
    {
        assert(t);
        if (b < 0)
            return 0;
        eval(t, b);
        bool f = (val >> (T)b) & (T)1;
        f ^= !t->ch[f];
        return getMinimum(t->ch[f], val, b - 1) | ((T)f << (T)b);
    }
    // Node t 以下の val 未満の要素数
    int countLessThan(Node *t, T val, int b = B - 1)
    {
        if (!t || b < 0)
            return 0;
        eval(t, b);
        bool f = (val >> (T)b) & (T)1;
        return (f && t->ch[0] ? t->ch[0]->cnt : 0) + countLessThan(t->ch[f], val, b - 1);
    }

  public:
    BinaryTrie() : root(nullptr) {}
```

```
    int size() const
    {
        return root ? root->cnt : 0;
    }
    bool empty() const
    {
        return !root;
    }
    // val を集合に1つ追加
    void insert(T val)
    {
        root = add(root, val);
    }
    // val を集合から1つ削除(なければRE)
    void erase(T val)
    {
        root = sub(root, val);
    }
    // 集合内の最大値
    T maximumElement(T mask = 0)
    {
        return getMinimum(root, ~mask);
    }
    // 集合内の最小値
    T minimumElement(T mask = 0)
    {
        return getMinimum(root, mask);
    }
    // 集合内で val 以上の要素の最小index
    int lowerBound(T val)
    {
        return countLessThan(root, val);
    }
    // 集合内で val より大きい要素の最小index
    int upperBound(T val)
    {
        return countLessThan(root, val + 1);
    }
    // k-th element(0-indexed) を取得
    T get(int k)
    {
        assert(0 <= k && k < size());
        return get(root, k);
    }
    // val の個数
    int count(T val)
    {
        if (!root)
            return 0;
        Node *t = root;
        for (int i = B - 1; i >= 0; i--)
        {
            eval(t, i);
            t = t->ch[(val >> (T)i) & (T)1];
            if (!t)
                return 0;
        }
        return t->cnt;
    }
    // 全ての要素を val で xor した値に変更
    void xorAllElements(T val)
    {
        if (root)
            root->lazy ^= val;
    }
};

```