线性代数

线性基

```cpp
std::vector<i64> get_linear_basis(std::vector<i64>& nums, int N = 63) {
    std::vector<i64> p(N + 1);
    auto insert = [&](i64 x) {
        for (int s = N;s >= 0;--s)if (x >> s & 1) {
            if (!p[s]) {
                p[s] = x;
                break;
            }
            x ^= p[s];
        }
        };
    for (auto& x : nums) insert(x);
    return p;
}

signed main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0), std::cout.tie(0);
    int n;std::cin >> n;
    std::vector<i64> nums(n);
    for (auto& x : nums)std::cin >> x;
    auto p = get_linear_basis(nums, 63);
    i64 ans = 0;
    for (int s = N;s >= 0;--s)
        ans = std::max(ans, ans ^ p[s]);
    std::cout << ans;
    return 0;
}
```

高斯消元法

设向量长度为 N（一般取 63），总数为 M，时间复杂度为 $\mathcal{O}(NM)$。

```cpp
struct LB { // Linear Basis
    using i64 = long long;
    const int BASE = 63;
    vector<i64> d, p;
    int cnt, flag;

    LB() {
        d.resize(BASE + 1);
        p.resize(BASE + 1);
        cnt = flag = 0;
    }
    bool insert(i64 val) {
        for (int i = BASE - 1; i >= 0; i--) {
            if (val & (1ll << i)) {
```

```
                if (!d[i]) {
                    d[i] = val;
                    return true;
                }
                val ^= d[i];
            }
        }
        flag = 1; //可以异或出0
        return false;
    }
    bool check(i64 val) { // 判断 val 是否能被异或得到
        for (int i = BASE - 1; i >= 0; i--) {
            if (val & (1ll << i)) {
                if (!d[i]) {
                    return false;
                }
                val ^= d[i];
            }
        }
        return true;
    }
    i64 ask_max() {
        i64 res = 0;
        for (int i = BASE - 1; i >= 0; i--) {
            if ((res ^ d[i]) > res) res ^= d[i];
        }
        return res;
    }
    i64 ask_min() {
        if (flag) return 0; // 特判 0
        for (int i = 0; i <= BASE - 1; i++) {
            if (d[i]) return d[i];
        }
    }
    void rebuild() { // 第k小值独立预处理
        for (int i = BASE - 1; i >= 0; i--) {
            for (int j = i - 1; j >= 0; j--) {
                if (d[i] & (1ll << j)) d[i] ^= d[j];
            }
        }
        for (int i = 0; i <= BASE - 1; i++) {
            if (d[i]) p[cnt++] = d[i];
        }
    }
    i64 kthquery(i64 k) { // 查询能被异或得到的第 k 小值, 如不存在则返回 -1
        if (flag) k--; // 特判 0, 如果不需要 0, 直接删去
        if (!k) return 0;
        i64 res = 0;
        if (k >= (1ll << cnt)) return -1;
        for (int i = BASE - 1; i >= 0; i--) {
            if (k & (1LL << i)) res ^= p[i];
        }
        return res;
```

```
    }
    void Merge(const LB &b) { // 合并两个线性基
        for (int i = BASE - 1; i >= 0; i--) {
            if (b.d[i]) {
                insert(b.d[i]);
            }
        }
    }
};
```

三角形面积

行列式求面积

$$S=\frac{1}{2}\begin{vmatrix}1 & 1 & 1\\ x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{vmatrix}$$

```
int main(){
    float num[6];
    for(int i = 0; i < 6; i++)
        cin >> num[i];
    float sum = 0.0;
    sum = 0.5*(num[0]*num[3]+num[2]*num[5]+num[4]*num[1]-num[0]*num[5]-num[2]*num[1]-num[4]*num[3]);
    cout << "三角形的面积为: ";
    sum == 0 ? cout << "Impossible" : cout <<sum;
    return 0;
}
```

海伦公式

$$S=\frac{1}{4}\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}$$

```
p=(a+b+c)/2;
sum=sqrt(p*(p-a)*(p-b)*(p-c));
```