Standard Cool Library

Patricky

Southwestern Petrolumn University

September 2022

Contents

初

```
#!/bin/zsh
g++ $1 -std=c++23 -fsanitize=undefined -g -Wall -Wextra -Wfatal-errors -Os -DLOCAL -o ${1%%.*} && time ./${1%%.*}
```

```
BasedOnStyle: LLVM
UseTab: Never
IndentWidth: 4
DerivePointerAlignment: false
PointerAlignment: true
AlwaysBreakAfterReturnType: None
AlwaysBreakTemplateDeclarations: true
AlwaysBreakBeforeMultilineStrings: true
AlignOperands: true
AlignAfterOpenBracket: true
AlignConsecutiveBitFields: true
AlignConsecutiveMacros: true
ConstructorInitializerAllOnOneLineOrOnePerLine: true
AllowAllConstructorInitializersOnNextLine: false
BinPackArguments: false
BinPackParameters: false
IncludeBlocks: Regroup
```

快读

```
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
}

template <typename T>
bool read(T& v) {
    static char ch;
    while (ch != EOF && !isdigit(ch)) ch = nc();
    if (ch == EOF) return false;
    for (v = 0; isdigit(ch); ch = nc()) v = v * 10 + ch - '0';
    return true;
}

template <typename T>
void write(T p) {
    static int stk[70], tp;
    if (p == 0) { putchar('0'); return; }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

对拍

```
#!/bin/zsh
g++ -o r main.cpp -O2 -std=c++17
g++ -o std std.cpp -O2 -std=c++17
while true; do
    python gen.py > in
    ./std < in > stdout
    ./r < in > out
    if test $? -ne 0; then
        exit 0
    fi
    if diff stdout out; then
        printf "AC\n"
    else
        printf "GG\n"
        exit 0
    fi
done
```

数据结构

线段树

```cpp
template <class S, S (*op)(S, S), S (*e)()>
class SegTree {
    const int n, siz;
    vector<int> node;
    vector<S> tr;
    void pushup(int u) { tr[u] = op(tr[u << 1], tr[u << 1 | 1]); }
    S get(int u, int l, int r, int ql, int qr) {
        if (qr <= l or r <= ql) { return e(); }
        if (ql <= l and r <= qr) { return tr[u]; }

        int m = (l + r) >> 1;
        return op( get(u << 1, l, m, ql, qr), get(u << 1 | 1, m, r, ql, qr));
    }
    template <class F>
    int lower_bound(int u, int l, int r, F &&check) {
        if (r - l == 1) { return l; }
        int m = (l + r) >> 1;
        if (check(tr[u << 1])) {
            return lower_bound(u << 1, l, m, check);
        } else {
            return lower_bound(u << 1 | 1, m, r, check);
        }
    }
    template <class F>
    int upper_bound(int u, int l, int r, F &&check) {
        if (r - l == 1) { return l; }
        int m = (l + r) >> 1;
        if (check(tr[u << 1 | 1])) {
            return lower_bound(u << 1 | 1, m, r, check);
        } else {
            return lower_bound(u << 1, l, m, check);
        }
    }
public:
    SegTree(int n) : SegTree(vector<S>(n, e())) { }
    SegTree(const vector<S> &a): n(a.size()), siz(4 << __lg(n)), node(n), tr(siz) {
        function<void(int, int, int)> build = [&](int u, int l, int r) {
            if (r - l == 1) { node[l] = u; tr[u] = a[l]; return ; }
            int m = (l + r) >> 1; build(u << 1, l, m); build(u << 1 | 1, m, r); pushup(u);
        };
        build(1, 0, n);
    }
    void set(int p, const S &v) { assert(0 <= p < n); p = node[p]; tr[p] = v; while (p >>= 1) { pushup(p); } }
    S get(int p) { assert(0 <= p < n); return tr[node[p]]; }
    S get(int l, int r) { assert(0 <= l <= r < n); return get(1, 0, n, l, r); }
    int lower_bound(function<bool(S)> &&check) { return lower_bound(1, 0, n, check); }
    int upper_bound(function<bool(S)> &&check) { return upper_bound(1, 0, n, check); }
};
```

lazy 线段树

```cpp
#define ls u << 1
#define rs ls | 1
template < class S, S (*op)(S, S), S (*e)(), class F, S (*mapping)(F, S), F (*merge)(F, F), F(*id)() >
// S 是元素类型
// op 是 S 和 S 的合并操作
// e 是 S 的单位元
// F 是懒标记
// mapping 是 F 对 S 的映射
// merge 是 F 和 F 的合并操作
// id 是 F 的单位元

/* 区间加 & 区间乘
 * struct S { Z sum; Z len; };
 * struct F { Z mul; Z add; };
 * S op(S a, S b) { return { a.sum + b.sum, a.len + b.len }; }
```

```cpp
 * S e() { return { 0, 0 }; }
 * S mapping(F f, S x) { return { x.sum * f.mul + f.add * x.len, x.len }; }
 * F merge(F f, F g) { return { f.mul * g.mul, f.mul * g.add + f.add }; }
 * F id() { return { 1, 0 }; }
 */
class LazySegTree {
    const int n, siz;
    vector<int> node;
    vector<S> tr;
    vector<F> tag;
    void pushup(int u) { tr[u] = op(tr[ls], tr[rs]); }
    void pushtag(int u, F t) {
        if (u >= siz) return ;
        tr[u] = mapping(t, tr[u]);
        tag[u] = merge(t, tag[u]);
    }
    void pushdown(int u) { pushtag(ls, tag[u]); pushtag(rs, tag[u]); tag[u] = id(); }
    S get(int u, int l, int r, int ql, int qr) {
        if (qr <= l or r <= ql) return e();
        if (ql <= l and r <= qr) return tr[u];
        pushdown(u); int m = (l + r) >> 1;
        return op( get(ls, l, m, ql, qr), get(rs, m, r, ql, qr));
    }
    template <class Func>
    int lower_bound(int u, int l, int r, Func &&check) {
        if (r - l == 1) { return l; }
        pushdown(u); int m = (l + r) >> 1;
        if (check(tr[ls])) {
            return lower_bound(ls, l, m, check);
        } else {
            return lower_bound(rs, m, r, check);
        }
    }
    template <class Func>
    int upper_bound(int u, int l, int r, Func &&check) {
        if (r - l == 1) { return l; }
        pushdown(u); int m = (l + r) >> 1;
        if (check(tr[rs])) {
            return lower_bound(rs, m, r, check);
        } else {
            return lower_bound(ls, l, m, check);
        }
    }
    void apply(int u, int l, int r, int ql, int qr, const F &v) {
        if (qr <= l or r <= ql) return ;
        if (ql <= l and r <= qr) { pushtag(u, v); return ; }
        pushdown(u); int m = (l + r) >> 1; apply(ls, l, m, ql, qr, v); apply(rs, m, r, ql, qr, v); pushup(u);
    }
public:
    LazySegTree(int n) : LazySegTree(vector<S>(n, e())) { }
    LazySegTree(const vector<S> &a): n(a.size()), siz(4 << __lg(n)), node(n), tr(siz), tag(siz, id()) {
        function<void(int, int, int)> build = [&](int u, int l, int r) {
            if (r - l == 1) { node[l] = u; tr[u] = a[l]; return ; }
            int m = (l + r) >> 1; build(ls, l, m); build(rs, m, r); pushup(u);
        };
        build(1, 0, n);
    }
    void set(int p, const S &v) { assert(0 <= p < n); p = node[p]; tr[p] = v; while (p >>= 1) { pushup(p); } }
    S get(int p) { assert(0 <= p < n); return tr[node[p]]; }
    S get(int l, int r) { assert(0 <= l <= r < n); return get(1, 0, n, l, r); }
    void apply(int l, int r, const F &t) { assert(0 <= l <= r < n); apply(1, 0, n, l, r, t); }
    int lower_bound(function<bool(S)> &&check) { return lower_bound(1, 0, n, check); }
    int upper_bound(function<bool(S)> &&check) { return upper_bound(1, 0, n, check); }
};
```

二维数点

$$\sum_{i=0}^{x}\sum_{j=0}^{y} S_{i,j}$$

$$\sum_{i=0}^{x}\sum_{j=0}^{y}[(i,j) \prec (x,y)], \text{ where } i \le x \wedge j \le y$$

```cpp
// x, y
array<int, 2> a[maxn];
// x, y, id, weight
array<int, 4> q[maxn << 2];
int n, m, t, ans[maxn];
int c[maxn * 5], tot;

int tr[maxn];
int qry(int i) { int ans{}; for (; i; i -= i & -i) { ans += tr[i]; } return ans; }
void mdf(int i, int v) { for (; i <= n; i += i & -i) { tr[i] += v; } }

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) { cin >> a[i][0] >> a[i][1]; a[i][0] += 1, a[i][1] += 1; c[++tot] = a[i][1]; }
    for (int i = 1; i <= m; ++i) { array<int, 4> qi{}; for (int j = 0; j < 4; ++j) { cin >> qi[j]; qi[j] += 1; }
        // 询问左上角的点记的是 y_1 - 1
        c[++tot] = qi[1] - 1;
        c[++tot] = qi[3];
        q[++t] = {qi[0] - 1, qi[1] - 1, i, 1}; // (x1 - 1, y1 - 1)
        q[++t] = {qi[0] - 1, qi[3], i, -1};    // (x1 - 1, y2)
        q[++t] = {qi[2], qi[1] - 1, i, -1};    // (x2, y1 - 1)
        q[++t] = {qi[2], qi[3], i, 1};         // (x2, y2)
    }

    sort(c + 1, c + 1 + tot); tot = unique(c + 1, c + 1 + tot) - (c + 1);
    for (int i = 1; i <= n; ++i) { a[i][1] = lower_bound(c + 1, c + 1 + tot, a[i][1]) - c; }
    for (int i = 1; i <= t; ++i) { q[i][1] = lower_bound(c + 1, c + 1 + tot, q[i][1]) - c; }
    sort(a + 1, a + 1 + n);
    sort(q + 1, q + 1 + t);

    for (int i = 1, j = 1; i <= t; ++i) {
        while (j <= n && a[j][0] <= q[i][0]) { mdf(a[j++][1], 1); }
        ans[q[i][2]] += q[i][3] * qry(q[i][1]);
    }
    for (int i = 1; i <= m; ++i) { cout << ans[i] << "\n"; }
    return 0;
}
```

图

DSU on tree

```cpp
void gsz(int u){sz[u]=1;for(int
  v:g[u]){dep[v]=dep[u]+1;xsum[v]^=xsum[u];gsz(v);sz[u]+=sz[v];if(sz[v]>sz[son[u]])son[u]=v;}}
void cal(int u,int p){rep(i,0,22)ckmax(ans[p],dep[u]+f[xsum[u]^bits[i]]);for(int v:g[u])cal(v,p);}
void add(int u){ckmax(f[xsum[u]],dep[u]);for(int v:g[u])add(v);}
void del(int u){f[xsum[u]]=-inf;          for(int v:g[u])del(v);}
void dsu(int u,int kp) {
  for(int v:g[u])if(v!=son[u])dsu(v,0);
  if(son[u]) dsu(son[u],1);
  ckmax(f[xsum[u]],dep[u]);
  rep(i,0,22)ckmax(ans[u],dep[u]+f[xsum[u]^bits[i]]);
  for(int v:g[u])if(v!=son[u])cal(v,u),add(v);
  if(!kp)del(u);
}
```

Dinic

```cpp
template<class T> struct Flow {
    const int n;
    struct Edge { int to; T cap; Edge(int _to, T _cap) : to(_to), cap(_cap) { } };
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> cur, h;
```

```
    Flow(int n) : n(n), g(n) {}

    bool bfs(int s, int t) {
        h.assign(n, -1);
        h[s] = 0;
        queue<int> q; q.push(s); while (q.size()) {
            const int u = q.front(); q.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 and h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) { return true; }
                    q.push(v);
                }
            }
        }
        return { };
    }

    T dfs(int u, int t, T f) {
        if (u == t) { return f; }
        auto r = f;
        for (int &i = cur[u]; i < static_cast<int>(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 and h[v] == h[u] + 1) {
                auto a = dfs(v, t, min(r, c));
                e[j].cap -= a; e[j ^ 1].cap += a; r -= a;
                if (r == 0) { return f; }
            }
        }
        return f - r;
    }

    void link(int u, int v, T c) {
        g[u].push_back(e.size()); e.emplace_back(v, c);
        g[v].push_back(e.size()); e.emplace_back(u, 0);
    }

    T run(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) { cur.assign(n, 0); ans += dfs(s, t, ::numeric_limits<T>::max()); }
        return ans;
    }
};

static constexpr int inf = 1E9;
```

mcmf

```
template <class Cap, class Cost>
class MCMF {
    static constexpr Cap INF = numeric_limits<Cap>::max();
    struct iedge { int v; Cap c; Cost f; iedge(int v, Cap c, Cost f) : v(v), c(c), f(f) {} };
    const int n;
    vector<iedge> e;
    vector<vector<int>> g;
    vector<Cost> h, dis;
    vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, INF);
        pre.assign(n, -1);
        using T = pair<Cost, int>;
        priority_queue<T, vector<T>, greater<T>> q; q.emplace(dist[s] = 0, s);
        while (!q.empty()) {
            Cost d = q.top().first;
            int u = q.top().second; q.pop();
            if (dis[u] != d) continue;
            for (int i : g[u]) {
                auto [v, c, f] = e[i];
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
```

```cpp
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    q.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != INF;
    }
public:
    struct Edge {
        int u, v;
        Cap flow;
        Cost cost;
        Edge(int u, int v, Cap flow, Cost cost) : u(u), v(v), flow(flow), cost(cost) { }
    };
    Flow(int n) : n(n), g(n) {}
    void addEdge(int u, int v, Cap cap, Cost cost) {
        g[u].push_back(e.size()); e.emplace_back(v, cap, cost);
        g[v].push_back(e.size()); e.emplace_back(u, 0, - cost);
    }
    pair<Cap, Cost> flow(int s, int t) {
        Cap flow = 0;
        Cost cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            Cap aug = INF;
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) { e[pre[i]].c -= aug; e[pre[i] ^ 1].c += aug; }
            flow += aug;
            cost += Cost(aug) * h[t];
        }
        return {flow, cost};
    }
    vector<Edge> edges() const {
        int m = e.size(); vector<Edge> res; res.reserve(m / 2);
        for (int i = 0; i < m; i += 2) { res.emplace_back(e[i ^ 1].v, e[i].v, e[i ^ 1].c, Cost(e[i ^ 1].c) * e[i].f); }
        return res;
    }
};
```

2SAT

```cpp
struct TwoSat {
    int n;
    vector<vector<int>> e;
    vector<bool> ans;

    TwoSat(int _n) : n(_n * 2), e(_n * 2), ans(_n) {};

    void addClause(int u, bool x, int v, bool y) {
        e[u << 1 | !x].push_back(v << 1 | y);
        e[v << 1 | !y].push_back(u << 1 | x);
    }

    const vector<bool>* run() {
        vector<int> dfn(n), low(n), id(n), stk;
        vector<bool> ins(n);
        int ts = 0, cnt = 0, y = 0;
        auto tarjan = y_combinator([&](auto tarjan, int u) -> void {
            dfn[u] = low[u] = ++ ts;
            stk.push_back(u), ins[u] = 1;
            for (int &v : e[u]) { if (!dfn[v]) { tarjan(v);
                    low[u] = min(low[u], low[v]);
                } else if (ins[v]) {
                    low[u] = min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                cnt ++; do { y = stk.back(); stk.pop_back(); ins[y] = 0, id[y] = cnt; } while (y != u);
            }
```

```
        });

        for (int i = 0; i < n; ++i) { if (!dfn[i]) { tarjan(i); } }
        for (int i = 0; i < n / 2; ++i) { if (id[i << 1] == id[i << 1 | 1]) { return nullptr; }
            ans[i] = id[i << 1] > id[i << 1 | 1];
        }
        return & ans;
    }
};
```

智力题

找出数组中仅出现一次的数。

其他数都出现 3 次。

$$\begin{cases} x' = x\bar{y}\bar{z} + \bar{x}yz = ? \\ y' = \bar{x}\bar{y}z + \bar{x}y\bar{z} = \bar{x}(y \oplus z) \end{cases}$$

```
int x{}, y{};
for (int z : A) {
  tie(x, y) = tuple{ (x & ~y & ~z) | (~x & y & z), ~x & (y ^ z) };
}
return y;
```

子集和

每一个元素在 2^n 个子集出现 2^{n-1} 次，因此答案为 $2^{n-1} \times \sum_{i=1}^{n} a_i$.

子段和

考虑每一个元素前驱后继覆盖到的区间数量，两部分独立，因此答案为 $\sum_{i=1}^{n} i \times (n - i) \times a_i$

子段最值和

从目标来看，任何子段的最值至多有 n 种取值，因此考虑每个 a_i 对哪些区间产生了贡献：答案明显是两边的 NGE，可使用单调栈。

```
long long ans { };

auto calc = [&](auto cmp, int sign) {
    vector stk(n + 1, 0), l(stk), r(stk);
    int top { };
    for (int i : ranges::iota_view(1, n + 1)) {
        while (top and cmp(a[i], a[stk[top]])) {
            r[stk[top --]] = i - 1;
        }
        l[i] = stk[top] + 1;
        stk[++top] = i;
    }
    while (top) { r[stk[top --]] = n; }
    for (int i : ranges::iota_view(1, n + 1)) {
        ans += sign * static_cast<long long>(i - l[i] + 1) * (r[i] - i + 1) * a[i];
    }
};
```

子段 gcd 和

每个 gcd 的贡献分为左右两部分实际上不如直接向右，因为有 $\gcd(x, y) \le \min(x, y)$ 这样的单调性，只需记下每一个 gcd 的出现次数。

```
#include <bits/stdc++.h>

int main() {
```

```cpp
    ios::sync_with_stdio(not cin.tie(nullptr));

    int n;
    cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }

    long long ans = 0;
    map<int, int> mp, nmp;
    for (int i = 1; i <= n; i++) {
        nmp.clear(); nmp[a[i]] = 1;
        for (auto [k, v] : mp) { nmp[__gcd(a[i], k)] += v; }
        for (auto [k, v] : nmp) { ans += static_cast<long long>(k) * v; }
        mp = move(nmp); // mp = nmp
    }

    cout << ans << "\n";

    return 0;
}
```

最少修改多少数使得原数组非严格递增

考虑贪心，$n - \operatorname*{LIS}_{i=1}^{n}\{a_i\}$ 即为所求。其中 LIS 非严格。

最少修改多少数使得原数组严格递增

如果两数大小关系不小于间隔距离，那么这两个点可以保留。形式化地说，需要满足：

$$a_j - a_i \geq j - i \Rightarrow a_j - j \geq a_i - i$$

于是 $n - \operatorname*{LIS}_{i=1}^{n}\{a_i - i\}$ 即为所求。

最大子段积

需要考虑负数，因此维护两个最值。

```cpp
long long m, M, ans;
m = M = ans = a[0];

for (int i : a | views::drop(1)) {
    tie(m, M) = minmax({ i, i * m, i * M });
    ans = max(ans, M);
}
```

一些题

对区间的异或依然是区间吗

n 个节点的树，给定边权为两点点权异或和 $w_{u,v} = w_u \oplus w_v$。每个点权的取值范围 $l_i \leq w_i \leq r_i$，求满足条件的 w_i 的数量。

- $n \in [1, 10^5], l_i, r_i \in [1, 2^{30}]$

确定任一点即可确定整棵树，不妨以根为中心。如果现在根节点是 $x \in [l_0, r_0]$，需要考虑对于其他节点来说是否有：

$$w_i \in [l_i, r_i] \overset{?}{\Rightarrow} w_i \oplus x \in [l_i, r_i]$$

等号成立，当且仅当 $[l, r]$ 中包含 2^k 个数并且低 k 位包含 $0 \sim 2^k - 1$。低位包含全部的数意为这些点是 Trie 的一棵子树. 用 Trie 标记出不合法的区间，再计算该子树的贡献即可（相当于对若干合法区间求交）。

```cpp
#include <bits/stdc++.h>

using ll = long long;

using namespace std;

enum { N = 30 * 100010 };
int n;

int head[N], cnt, u, v, w;
struct {
    int next, to, w;
}
edges[N << 1];
void addEdge(int u, int v, int w) {
    edges[++cnt] = { head[u], v, w };
    head[u] = cnt;
}

// Trie 部分. T tag {0/1/2} 表示 {2/1/0} 个儿子合法
int son[N][2], tot;
int T[N];
int newNode() {
    ++tot;
    T[tot] = son[tot][0] = son[tot][1] = 0;
    return tot;
}

int trie(int u, int b, int l, int r, int w) {
    // 已经不合法/这颗子树覆盖 2 ^ bits 个树
    // 即 r - l = 2 ^ (b + 1) - 1 下同
    if (T[u] == 2 || r - l == ~-(1 << -~b)) {
        return T[u];
    }

    for (int v: { 0, 1 }) {
        if (!son[u][v]) {
            son[u][v] = newNode();
        }
    }

    // l, r 第 b 位相同
    if (int m = 1 << b;
        (l & m) == (r & m)) {
        // 当前的权与 l 第 b 位相同
        // 不相同的那一部分 l, r 也管不了直接标记为无用
        bool st = (l & m) ^ (w & m);
        T[son[u][!st]] = 2;
        // m = 2 ^ b. %m 其实就是 & ( m - 1 )
        T[u] = trie(son[u][st], b - 1, l % m, r % m, w) == 2 ? 2 : 1;
    } else {
        bool st = w & m;
        // b 位是 1/0 时子树的情况
        int lft = trie(son[u][st], b - 1, l % m, m - 1, w);
        int rgt = trie(son[u][!st], b - 1, 0, r % m, w);

        if (lft == 2 && rgt == 2) {
            T[u] = 2;
        } else if (lft != 2 || rgt != 2) {
            T[u] = 1;
        }
    }

    return T[u];
}

int L[N], R[N];
void dfs(int u, int p, int w) {
    trie(1, 29, L[u], R[u], w);
    for (int i = head[u]; i; i = edges[i].next) {
        if (int v = edges[i].to; v != p) {
```

```
            dfs(v, u, w ^ edges[i].w);
        }
    }
}

ll ans {};
void getAns(int u, int b) {
    if (!T[u]) {
        // 整棵树都可以用
        ans += 1 LL << -~b;
    } else if (T[u] == 1) {
        // 只有一边能用
        for (int v: { 0, 1 }) {
            if (T[son[u][v]] != 2) {
                getAns(son[u][v], b - 1);
            }
        }
    }
}

int main() {
    scanf("%d", & n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d", L + i, R + i);
    }

    for (int i = 1; i < n; ++i) {
        scanf("%d%d%d", & u, & v, & w);
        addEdge(u, v, w);
        addEdge(v, u, w);
    }

    newNode();
    dfs(1, 0, 0);
    getAns(1, 29);
    printf("%lld", ans);

    return 0 ^ 0;
}
```

多次询问某区间内所有区间的异或和。

按位考虑，对于每一位来说，其贡献 s_i 为 1 的位。于是答案为「前缀异或和数组中两两异或再求和的值」，也就是只有 01 之间才有贡献，于是每一段的贡献为：

$$2^i \times \text{cnt}_1 \times \text{cnt}_0$$

```
int s[bits][maxn];

for (int i = 1, x; i <= n; ++i) {
    std::cin >> x;
    for (int j = 0; j < bits; ++j) {
        s[j][i] = s[j][i - 1] ^ (x >> j & 1);
    }
}

for (int j = 0; j < bits; ++j) {
    for (int i = 1; i <= n; ++i) {
        s[j][i] += s[j][i - 1];
    }
}

l -= 1;

ll ans {};
for (int j = 0; j < bits; ++j) {
    int o = s[j][r] - s[j][l - 1], z = (r - l + 1) - o;
    ans = (ans + 1 LL * o * z % mod * (1 LL << j) % mod) % mod;
}
```

数学

Z

```cpp
static constexpr int P = 1000000007;

// assume -P <= x < 2P
int norm(int x) { return x >= P ? x - P : x < 0 ? x + P : x; }
template <class T> T power(T a, int b) {
  T res = 1;
  for (; b; b /= 2, a *= a) {
    if (b % 2) { res *= a; }
  }
  return res;
}
struct Z {
  int x;
  Z(int x = 0) : x(norm(x)) {}
  Z(long long x) : x(norm(x % P)) {}
  int val() const { return x; }
  friend ostream &operator<<(ostream &os, const Z &x) {
    return os << x.val();
  }
  Z operator-() const { return Z(norm(P - x)); }
  Z inv() const { assert(x != 0); return power(*this, P - 2); }
  Z &operator*=(const Z &rhs) { x = static_cast<long long>(x) * rhs.x % P; return *this; }
  Z &operator+=(const Z &rhs) { x = norm(x + rhs.x); return *this; }
  Z &operator-=(const Z &rhs) { x = norm(x - rhs.x); return *this; }
  Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
  friend Z operator*(const Z &lhs, const Z &rhs) { Z res = lhs; res *= rhs; return res; }
  friend Z operator+(const Z &lhs, const Z &rhs) { Z res = lhs; res += rhs; return res; }
  friend Z operator-(const Z &lhs, const Z &rhs) { Z res = lhs; res -= rhs; return res; }
  friend Z operator/(const Z &lhs, const Z &rhs) { Z res = lhs; res /= rhs; return res; }
};
```

Poly

```cpp
vector<int> rev;
vector<Z> roots{0, 1};
void dft(vector<Z> &a) {
  int n = a.size();

  if (int(rev.size()) != n) {
    int k = __builtin_ctz(n) - 1;
    rev.resize(n);
    for (int i = 0; i < n; i++) { rev[i] = rev[i >> 1] >> 1 | (i & 1) << k; }
  }

  for (int i = 0; i < n; i++) { if (rev[i] < i) { swap(a[i], a[rev[i]]); } }
  if (int(roots.size()) < n) {
    int k = __builtin_ctz(roots.size());
    roots.resize(n);
    while ((1 << k) < n) {
      Z e = power(Z(3), (P - 1) >> (k + 1));
      for (int i = 1 << (k - 1); i < (1 << k); i++) { roots[2 * i] = roots[i]; roots[2 * i + 1] = roots[i] * e; }
      k++;
    }
  }
  for (int k = 1; k < n; k *= 2) {
    for (int i = 0; i < n; i += 2 * k) {
      for (int j = 0; j < k; j++) {
        Z u = a[i + j], v = a[i + j + k] * roots[k + j];
        a[i + j] = u + v; a[i + j + k] = u - v;
      }
    }
  }
}
void idft(vector<Z> &a) {
  int n = a.size();
  reverse(a.begin() + 1, a.end());
```

```
  dft(a);
  Z inv = (1 - P) / n;
  for (int i = 0; i < n; i++) { a[i] *= inv; }
}
struct Poly {
  vector<Z> a;
  Poly() {}
  Poly(const vector<Z> &a) : a(a) {}
  Poly(const initializer_list<Z> &a) : a(a) {}
  int size() const { return a.size(); }
  void resize(int n) { a.resize(n); }
  Z operator[](int idx) const { if (idx < 0 || idx >= size()) { return 0; } return a[idx]; }
  Z &operator[](int idx) { return a[idx]; }
  Poly mulxk(int k) const { auto b = a; b.insert(b.begin(), k, 0); return Poly(b); }
  Poly modxk(int k) const { k = min(k, size()); return Poly(vector<Z>(a.begin(), a.begin() + k)); }
  Poly divxk(int k) const { if (size() <= k) { return Poly(); } return Poly(vector<Z>(a.begin() + k, a.end())); }
  friend Poly operator+(const Poly &a, const Poly &b) {
    vector<Z> res(max(a.size(), b.size()));
    for (int i = 0; i < int(res.size()); i++) { res[i] = a[i] + b[i]; }
    return Poly(res);
  }
  friend Poly operator-(const Poly &a, const Poly &b) {
    vector<Z> res(max(a.size(), b.size()));
    for (int i = 0; i < int(res.size()); i++) { res[i] = a[i] - b[i]; }
    return Poly(res);
  }
  friend Poly operator*(Poly a, Poly b) {
    if (a.size() == 0 || b.size() == 0) { return Poly(); }
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz *= 2;
    a.a.resize(sz); b.a.resize(sz);
    dft(a.a); dft(b.a);
    for (int i = 0; i < sz; ++i) { a.a[i] = a[i] * b[i]; }
    idft(a.a);
    a.resize(tot); return a;
  }
  friend Poly operator*(Z a, Poly b) {
    for (int i = 0; i < int(b.size()); i++) { b[i] *= a; }
    return b;
  }
  friend Poly operator*(Poly a, Z b) {
    for (int i = 0; i < int(a.size()); i++) { a[i] *= b; }
    return a;
  }
  Poly &operator+=(Poly b) { return (*this) = (*this) + b; }
  Poly &operator-=(Poly b) { return (*this) = (*this) - b; }
  Poly &operator*=(Poly b) { return (*this) = (*this) * b; }
  Poly deriv() const {
    if (a.empty()) { return Poly(); }
    vector<Z> res(size() - 1);
    for (int i = 0; i < size() - 1; ++i) { res[i] = (i + 1) * a[i + 1]; }
    return Poly(res);
  }
  Poly integr() const {
    vector<Z> res(size() + 1);
    for (int i = 0; i < size(); ++i) { res[i + 1] = a[i] / (i + 1); }
    return Poly(res);
  }
  Poly inv(int m) const {
    Poly x{a[0].inv()};
    int k = 1;
    while (k < m) { k *= 2; x = (x * (Poly{2} - modxk(k) * x)).modxk(k); }
    return x.modxk(m);
  }
  Poly log(int m) const { return (deriv() * inv(m)).integr().modxk(m); }
  Poly exp(int m) const {
    Poly x{1};
    int k = 1;
    while (k < m) { k *= 2; x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k); }
    return x.modxk(m);
  }
```

```
  Poly pow(int k, int m) const {
    int i = 0;
    while (i < size() && a[i].val() == 0) { i++; }
    if (i == size() || 1LL * i * k >= m) { return Poly(vector<Z>(m)); }
    Z v = a[i];
    auto f = divxk(i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
  }
  Poly sqrt(int m) const {
    Poly x{1}; int k = 1;
    while (k < m) { k *= 2; x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2); }
    return x.modxk(m);
  }
  Poly mulT(Poly b) const {
    if (b.size() == 0) { return Poly(); }
    int n = b.size();
    reverse(b.a.begin(), b.a.end());
    return ((*this) * b).divxk(n - 1);
  }
  vector<Z> eval(vector<Z> x) const {
    if (size() == 0) { return vector<Z>(x.size(), 0); }
    const int n = max(int(x.size()), size());
    vector<Poly> q(4 * n);
    vector<Z> ans(x.size());
    x.resize(n);
    function<void(int, int, int)> build = [&](int p, int l, int r) {
      if (r - l == 1) {
        q[p] = Poly{1, -x[l]};
      } else {
        int m = (l + r) / 2; build(2 * p, l, m); build(2 * p + 1, m, r); q[p] = q[2 * p] * q[2 * p + 1];
      }
    };
    build(1, 0, n);
    function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
      if (r - l == 1) { if (l < int(ans.size())) { ans[l] = num[0]; }
      } else {
        int m = (l + r) / 2;
        work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
        work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
      }
    };
    work(1, 0, n, mulT(q[1].inv(n)));
    return ans;
  }
};
```

杂项

```
template <class T, class... Args> auto multivector(size_t n, Args &&...args) {
  if constexpr (sizeof...(args) == 1) {
    return vector<T>(n, args...);
  } else {
    return vector(n, multivector<T>(args...));
  }
}

template <class Fun> struct y_combinator_result {
  Fun fun_;
  template <class T>
  explicit y_combinator_result(T &&fun) : fun_(forward<T>(fun)) {}
  template <class... Args> decltype(auto) operator()(Args &&...args) {
    return fun_(ref(*this), forward<Args>(args)...);
  }
};

template <class Fun> decltype(auto) y_combinator(Fun &&fun) {
  return y_combinator_result<decay_t<Fun>>(forward<Fun>(fun));
}

template <typename... Args>
```

```cpp
void log(const Args &...args) { ((clog << args << ", "), ...); }

template <typename... Ts>
void dump(const tuple<Ts...> &tp) {
  apply([](const auto &...args) { ((cout << args << " "), ...); }, tp);
}
```

随机数

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
auto rd  = bind(uniform_real_distribution<double>(0, 1), rng);
auto rd2 = bind(uniform_int_distribution<int>(_1, _2),   rng);
```

浮点数比较

意义	写法
$a = b$	`fabs(a - b) < epsilon`
$a \neq b$	`fabs(a - b) > epsilon`
$a < b$	`a - b < - epsilon`
$a \leq b$	`a - b < epsilon`
$a > b$	`a - b > epsilon`
$a \geq b$	`a - b > - epsilon`

二维偏序排序规则

除第一维是 $\geq$ 外，其余情况还需满足可重。否则退化为普通情形，即直接按照符号（不带等号）排。

第一维	x
$<=$	默认
$<$	第二维逆序排序
$>=$	第一维逆序排序
$>$	两维都逆序排序

第二维	y
$<=$	默认
$<$	查询时使用 ‘query(x - 1)
$>=$	离散化时逆序排序
$>$	结合前两者

G

- ☐ 有需要开 `long long` 的地方吗?
- ☐ BFS 边界写对了吗?
- ☐ 调试信息都删干净了吗?