线段树进行区间异或 求和

对 [L, R] 区间上的每个数都异或上 val

```
#include <bits/stdc++.h>
using namespace std;
#define GL (k << 1)
#define GR (k << 1 | 1)
#define i64 long long
const int N = 30;
struct Segt
{ // #define GL (k << 1) // #define GR (k << 1 | 1)
    struct node
    {
        int l, r;
        int w[N], lazy; // 注意这里为了方便计算，w 只需要存位
    };
    vector<int> base;
    vector<node> t;
    Segt(vector<int> in) : base(in)
    {
        int n = in.size() - 1;
        t.resize(n * 4 + 1);
        auto build = [&](auto self, int l, int r, int k = 1)
        {
            t[k] = {l, r}; // 前置赋值
            if (l == r)
            {
                for (int i = 0; i < N; i++)
                {
                    t[k].w[i] = base[l] >> i & 1;
                }
                return;
            }
            int mid = (l + r) / 2;
            self(self, l, mid, GL);
            self(self, mid + 1, r, GR);
            pushup(k);
        };
        build(build, 1, n);
    }
    void pushdown(node &p, int lazy)
    { /* 【在此更新下递函数】 */
        int len = p.r - p.l + 1;
        for (int i = 0; i < N; i++)
        {
            if (lazy >> i & 1)
            { // 即 p.w = (p.r - p.l + 1) - p.w;
                p.w[i] = len - p.w[i];
            }
```

```
        }
        p.lazy ^= lazy;
    }
    void pushdown(int k)
    { // 【不需要动】
        if (t[k].lazy == 0)
            return;
        pushdown(t[GL], t[k].lazy);
        pushdown(t[GR], t[k].lazy);
        t[k].lazy = 0;
    }
    void pushup(int k)
    {
        auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函数】 */
            for (int i = 0; i < N; i++)
            {
                p.w[i] = l.w[i] + r.w[i]; // 即 p.w = l.w + r.w;
            }
        };
        pushup(t[k], t[GL], t[GR]);
    }
    void modify(int l, int r, int val, int k = 1)
    { // 区间修改 对区间[L, R]内的每个数都异或上 val
        if (l <= t[k].l && t[k].r <= r)
        {
            pushdown(t[k], val);
            return;
        }
        pushdown(k);
        int mid = (t[k].l + t[k].r) / 2;
        if (l <= mid)
            modify(l, r, val, GL);
        if (mid < r)
            modify(l, r, val, GR);
        pushup(k);
    }
    i64 ask(int l, int r, int k = 1)
    { // 区间求和
        if (l <= t[k].l && t[k].r <= r)
        {
            i64 ans = 0;
            for (int i = 0; i < N; i++)
            {
                ans += t[k].w[i] * (1LL << i);
            }
            return ans;
        }
        pushdown(k);
        int mid = (t[k].l + t[k].r) / 2;
        i64 ans = 0;
        if (l <= mid)
            ans += ask(l, r, GL);
        if (mid < r)
            ans += ask(l, r, GR);
```

```
        return ans;
    }
};
void solve()
{
    int n;
    cin >> n;
    vector<int> a(n + 5);
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    Segt tr(a);
    int q;
    cin >> q;
    for (int qk = 1; qk <= q; qk++)
    {
        int op, L, R, x;
        cin >> op;
        if (op == 1)
        {
            cin >> L >> R;
            cout << tr.ask(L, R) << endl;
        }
        else
        {
            cin >> L >> R >> x;
            tr.modify(L, R, x);
        }
    }
}

signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int t = 1;
    while (t--)
        solve();
    return 0;
}
```

线段树区间位翻转求和

对 区间 [L, R] 上的每个数根据 val 进行位翻转

```
#include <bits/stdc++.h>
#define GL (k << 1)
#define GR (k << 1 | 1)
#define i64 long long
using namespace std;
template<class T> struct Segt_{
```

```
struct node{
    int pl, pr;
    T w;
    bool lazy;
};
vector<T> w;
vector<node> t;
Segt_(){}
void init(vector<int> in){
    int n = in.size() - 1;
    w.resize(n * 4 + 1);
    for (int i = 0; i <= n; i++){
        w[i] = in[i];
    }
    t.resize(n * 4 + 1);
    build(1, n);
}
void pushdown(node &p, bool lazy = 1){
    p.w = (p.pr - p.pl + 1) - p.w;
    p.lazy ^= lazy;
}
void pushup(node &p, node &pl, node &pr){
    p.w = pl.w + pr.w;
}
void pushdown(int k){
    if (t[k].lazy == 0) return;
    pushdown(t[GL]), pushdown(t[GR]);
    t[k].lazy = 0;
}
void pushup(int k){
    pushup(t[k], t[GL], t[GR]);
}
void build(int pl, int pr, int k = 1){
    if (pl == pr){
        t[k] = {pl, pr, w[pl], 0};
        return ;
    }
    t[k] = {pl, pr};
    int mid = (pl + pr) / 2;
    build(pl, mid, GL);
    build(mid + 1, pr, GR);
    pushup(k);
}
void reverse(int pl, int pr, int k = 1){
    if (pl <= t[k].pl && t[k].pr <= pr){
        pushdown(t[k], 1);
        return;
    }
    pushdown(k);
    int mid = (t[k].pl + t[k].pr) / 2;
    if (pl <= mid) reverse(pl, pr, GL);
    if (mid < pr) reverse(pl, pr, GR);
    pushup(k);
}
```

```
    T ask(int L, int R, int k = 1){
        if (L <= t[k].pl && t[k].pr <= R){
            return t[k].w;
        }
        pushdown(k);
        int mid = (t[k].pl + t[k].pr) / 2;
        T ans = 0;
        if (L <= mid) ans += ask(L, R, GL);
        if (mid < R) ans += ask(L, R, GR);
        return ans;
    }
};
/*
5
1 2 3 4 5
对 [1, 3] 区间进行 x = 3 (0011) 的翻转， 则：
1 (0001) -> 2 (0010)
2 (0010) -> 1 (0001)
3 (0011) -> 0 (0000)
*/
signed main(){
    int n;
    cin >> n;
    vector in(20, vector<int>(n + 1));
    Segt_<i64> segt[20];
    for (int i = 1, x; i <= n; i++){
        cin >> x;
        for (int bit = 0; bit < 20; bit++){
            in[bit][i] = x >> bit & 1;
        }
    }
    for (int i = 0; i < 20; i++){
        segt[i].init(in[i]);
    }
    int m, op;
    for (cin >> m; m; m--){
        cin >> op;
        // 操作一：求 [L, R] 区间上的和
        if (op == 1){
            int L, R;
            i64 ans = 0;
            cin >> L >> R;
            for (int i = 0; i < 20; i++){
                ans += segt[i].ask(L, R) * (1LL << i);
            }
            cout << ans << endl;
        }
        // 操作二：对[L, R] 上每一位数字的二进制根据给出的x进行翻转
        else{
            int L, R, val;
            cin >> L >> R >> val;
            for (int i = 0; i < 20; i++){
                if (val >> i & 1){
                    segt[i].reverse(L, R);
```

```
                }
            }
        }
    }
}
```

数组数组

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
using namespace std;

template <class T>
struct Fenwick
{
    int n;
    vector<T> t;

    Fenwick(T n) : n(n) { t.assign(n + 1, T{}); }

    void add(int x, const T &v)
    {
        for (int i = x; i <= n; i += i & -i)
        {
            t[i] += v;
        }
    }

    T sum(int x)
    {
        assert(x >= 0);
        int res = 0;
        for (int i = x; i; i -= i & -i)
        {
            res += t[i];
        }
        return res;
    }

    T range(int L, int R)
    {
        return sum(R) - sum(L - 1);
    }

    int select(const T &k) // 小于等于 k 的最大位置
    {
        int x = 0;
        T cur{};
        for (int i = 1 << __lg(n); i; i /= 2)
```

```
        {
            if (x + i <= n && cur + t[x + i] <= k)
            {
                x += i;
                cur += t[x];
            }
        }
        return x;
    }
};
signed main()
{
    Fenwick<int> tr(10);
    tr.add(5, 2);
    tr.add(7, 2);
    cout << tr.select(3) << endl;
    return 0;
}
```

主席树

```
#include <iostream>
#include <vector>
using namespace std;

struct PresidentTree
{
    struct node
    {
        int l, r;
        int cnt;
    };
    int cntNodes{}, n{};
    vector<int> root;
    vector<node> tr;

    PresidentTree(int n)
    {
        cntNodes = 0;
        this->n = n;
        root.resize(n << 7 | 1, 0);
        tr.resize(n << 7 | 1);
        build(root[0], 1, n);
    }

    void build(int &u, int l, int r)
    {                   // 建空树
        u = ++cntNodes; // 动态开点
        if (l == r)
            return;
```

```
    int mid = (l + r) >> 1;
    build(tr[u].l, l, mid);
    build(tr[u].r, mid + 1, r);
}

void modify(int &u, int v, int l, int r, int x)
{
    u = ++cntNodes;
    tr[u] = tr[v];
    tr[u].cnt++;
    if (l == r)
        return;
    int mid = (l + r) / 2;
    if (x <= mid)
        modify(tr[u].l, tr[v].l, l, mid, x);
    else
        modify(tr[u].r, tr[v].r, mid + 1, r, x);
}

void modify(int cur, int pre, int x)
{
    modify(root[cur], root[pre], 1, n, x);
}

int kth(int u, int v, int l, int r, int k)
{
    if (l == r)
        return l;
    int res = tr[tr[v].l].cnt - tr[tr[u].l].cnt;
    int mid = (l + r) / 2;
    if (k <= res)
        return kth(tr[u].l, tr[v].l, l, mid, k);
    else
        return kth(tr[u].r, tr[v].r, mid + 1, r, k - res);
}

int kth(int l, int r, int k) // 区间[L, R] 内第 k 大的数是多少
{
    if (l > r)
        return 0;
    return kth(root[l - 1], root[r], 1, n, k);
}

int ask(int u, int v, int l, int r, int k)
{
    if (l == r)
        return tr[v].cnt - tr[u].cnt;
    int mid = (l + r) / 2;
    int ans = 0;
    if (k <= mid)
        ans += ask(tr[u].l, tr[v].l, l, mid, k);
    else
    {
        ans += tr[tr[v].l].cnt - tr[tr[u].l].cnt;
```

```
            ans += ask(tr[u].r, tr[v].r, mid + 1, r, k);
        }
        return ans;
    }

    int ask(int l, int r, int k) // 区间[L, R] 内比 k 小的数有几个
    {
        if (l > r)
            return 0;
        return ask(root[l - 1], root[r], 1, n, k);
    }
};
```