Standard Code Library

Part2 - Data Structure

Jiangxi Normal University

HeartFireY, eroengine, yezzz

November 19, 2022

Standard Code Library

HeartFireY, eroengine, yezzz

Jiangxi Normal University

November 19, 2022

Contents

数据结构

[TOC]

1. 离散化

数组版

```cpp
for(int i = 1; i <= n; i++) std::cin >> a[i], b[i] = a[i];
std::sort(a + 1, a + 1 + n);
len = std::unique(a + 1, a + 1 + n) - a - 1;
auto query_pos = [&](int x) { return std::lower_bound(a + 1, a + 1 + len, x) - a; };
```

向量版

```cpp
vector<int> a;
std::sort(a.begin(),a.end());
a.erase(unique(a.begin(),a.end()),a.end());
auto query_pos = [&](int x) { return lower_bound(a.begin(), a.end(), c) - a.begin() + 1;
//查找下标从 1 开始
}
```

2. 并查集

路径压缩 + 按秩合并

```cpp
struct UnionFind {
    std::vector<int> par, rank, size;
    int c;
    UnionFind(int n) : par(n), rank(n, 0), size(n, 1), c(n) {
        for(int i = 0; i < n; ++i) par[i] = i;
    }
    int find(int i) { return (par[i] == i ? i : (par[i] = find(par[i]))); }
    bool same(int i, int j) { return find(i) == find(j); }
    int get_size(int i) { return size[find(i)]; }
    int count() { return c; }
    int merge(int i, int j) {
        if((i = find(i)) == (j = find(j))) return -1;
        else --c;
        if(rank[i] > rank[j]) std::swap(i, j);
        par[i] = j, size[j] += size[i];
        if(rank[i] == rank[j]) rank[j]++;
        return j;
    }
};
```

可回滚并查集

- 注意这个不是可持久化并查集
- 查找时不进行路径压缩
- 复杂度靠按秩合并解决

```cpp
namespace uf {
    int fa[maxn], sz[maxn];
    int undo[maxn], top;
    void init() { memset(fa, -1, sizeof fa); memset(sz, 0, sizeof sz); top = 0; }
    int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
    bool join(int x, int y) {
        x = findset(x); y = findset(y);
        if (x == y) return false;
        if (sz[x] > sz[y]) swap(x, y);
        undo[top++] = x;
        fa[x] = y;
        sz[y] += sz[x] + 1;
        return true;
    }
    inline int checkpoint() { return top; }
    void rewind(int t) {
        while (top > t) {
```

```cpp
                int x = undo[--top];
                sz[fa[x]] -= sz[x] + 1;
                fa[x] = -1;
            }
        }
}
```

3.ST 表

- 预处理: $O(n \log n)$
- 查询: $O(1)$

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

int n, q;

namespace ST{
    const int N = 2000010;
    int stmax[N][22], stmin[N][22], mn[N], a[N];

    void init(int n){
        mn[0] = -1;
        for (int i = 1; i <= n; i++){
            mn[i] = ((i & (i - 1)) == 0) ? mn[i - 1] + 1 : mn[i - 1];
            stmax[i][0] = stmin[i][0] = a[i];
        }
        for (int j = 1; j <= mn[n]; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++){
                stmax[i][j] = max(stmax[i][j - 1], stmax[i + (1 << (j - 1))][j - 1]);
                stmin[i][j] = min(stmin[i][j - 1], stmin[i + (1 << (j - 1))][j - 1]);
            }
    }

    inline int rmq_max(int L, int R){
        int k = mn[R - L + 1];
        return max(stmax[L][k], stmax[R - (1 << k) + 1][k]);
    }

    inline int rmq_min(int L, int R){
        int k = mn[R - L + 1];
        return min(stmin[L][k], stmin[R - (1 << k) + 1][k]);
    }
}

signed main(){
    cin >> n >> q;
    for(int i = 1; i <= n; i++) cin >> ST::a[i];
    ST::init(n);
    while(q--){
        int l, r; cin >> l >> r;
        cout << ST::rmq_max(l, r) << ' ' << ST::rmq_min(l, r) << endl;
    }
    return 0;
}
```

4. 树状数组 (Fenwick)

朴素树状数组

查询区间第 K 大需要权值树状数组!

```cpp
namespace Fenwick{
    int tree[N], len;
    #define lowbit(x) ((x) & (-x))
    inline void init(){
        memset(tree, 0, sizeof(tree));
        for(int i = 1, tmp = 0; i <= len; i++){
            tree[i] += a[i];
            tmp = i + lowbit(i);
```

```
            if(tmp <= len) tree[tmp] += tree[i];
        }
    }

    inline void update(int i, int x){
        for(int pos = i; pos <= len; pos += lowbit(pos)) tree[pos] += x;
    }

    inline int getsum(int i, int ans = 0){
        for(int pos = i; pos; pos -= lowbit(pos)) ans += tree[pos];
        return ans;
    }

    inline int query(int l, int r){ return getsum(r) - getsum(l - 1); }

    //* 查询区间第 K 大需要权值树状数组!
    int kth(int i){
        int cnt = 0, ret = 0;
        for(int i = log2(len); ~i; --i){
            ret += 1 << i;
            (ret >= len || cnt + tree[ret] >= i) ? (ret -= 1 << i) : (cnt += tree[ret]);
        }
        return ret + 1;
    }
}
```

区间加/区间求和

若维护序列 a 的差分数组 b，此时我们对 a 的一个前缀 r 求和，即 $\sum_{i=1}^{r} a_i$，由差分数组定义得 $a_i = \sum_{j=1}^{i} b_j$

进行推导

$$\sum_{i=1}^{r} a_i$$

$$=\sum_{i=1}^{r}\sum_{j=1}^{i} b_j$$

$$=\sum_{i=1}^{r} b_i \times (r-i+1)$$

$$=\sum_{i=1}^{r} b_i \times (r+1) - \sum_{i=1}^{r} b_i \times i$$

区间和可以用两个前缀和相减得到，因此只需要用两个树状数组分别维护 $\sum b_i$ 和 $\sum i \times b_i$，就能实现区间求和。

```
namespace Fenwick_Plus{
    #define lowbit(x) ((x) & (-x))
    #define MAXN $Array_Max_Sizes$
    int tree1[MAXN], tree2[MAXN], a[MAXN], len;
    //对两个树状数组进行更新
    inline void add(int i, int x){
        int x1 = i * x;
        for(int pos = i; pos <= len; pos += lowbit(pos)) tree1[pos] += x, tree2[pos] += x1;
    }
    //将区间加差分为两个前缀和
    inline void update(int l, int r, int x){
        add(l, x), add(r + 1, -x);
    }
    //对指定的树状数组求前 n 项和
    inline int getsum(int *tree, int i){
        int sum = 0;
        for(int pos = i; pos; pos -= lowbit(pos)) sum += tree[i];
        return sum;
    }
    //区间和查询
    inline int query(int l, int r){
        return (r + 1) * getsum(tree1, r) - l * getsum(tree1, l - 1) - (getsum(tree2, r) - getsum(tree2, l - 1));
    }
```

```
}
```

5. 线段树模板

(1). 朴素线段树

区间和：区间修改/单点修改/单点查询/区间查询

```
#define SEGRG 1, 1, n //! 此处定义范围，注意 n 取值

const int N = 1e5 + 10;

namespace SegmentTree{
    #define ls rt << 1
    #define rs rt << 1 | 1
    #define lson rt << 1, l, mid
    #define rson rt << 1 | 1, mid + 1, r
    int len, q, tree[N << 2], lazy[N << 2];
    inline void push_up(int rt) { tree[rt] = tree[ls] + tree[rs]; }

    inline void push_down(int rt, int m){
        if(!lazy[rt]) return;
        lazy[ls] += lazy[rt], lazy[rs] += lazy[rt];
        tree[ls] += lazy[rt] * (m - (m >> 1));
        tree[rs] += lazy[rt] * (m >> 1);
        lazy[rt] = 0;
    }

    static void build(int rt, int l, int r){
        tree[rt] = lazy[rt] = 0;
        if(l == r){
            tree[rt] = 0; //!build leaf_node here
            return;
        }
        int mid = l + r >> 1;
        build(lson), build(rson);
        push_up(rt);
    }

    static void update_part(int rt, int l, int r, int L, int R, int val){
        if(l >= L && r <= R){
            lazy[rt] += val, tree[rt] += (r - l + 1) * val;
            return;
        }
        int mid = l + r >> 1;
        push_down(rt, r - l + 1);
        if(mid >= L) update_part(lson, L, R, val);
        if(mid < R) update_part(rson, L, R, val);
        push_up(rt);
    }

    static void update_point(int rt, int l, int r, int pos, int val){
        if(l == r){
            tree[rt] += val;
            return;
        }
        push_down(rt, r - l + 1);
        int mid = l + r >> 1;
        if(mid >= pos) update_point(lson, pos, val);
        else update_point(rson, pos, val);
        push_up(rt);
    }

    static int query(int rt, int l, int r, int L, int R){
        if(l >= L && r <= R) return tree[rt];
        int mid = l + r >> 1, ans = 0;
        push_down(rt, r - l + 1);
        if(mid >= L) ans += query(lson, L, R);
        if(mid < R) ans += query(rson, L, R);
        return ans;
    }
```

```
}
```

(2). 动态开点线段树

```cpp
namespace SegmentTree{
    const int N = 3e5 + 10;
    int tree[N << 2], lson[N << 2], rson[N << 2], lazy[N << 2], tot = 0, root = 0;

    inline void push_up(int rt){ tree[rt] = tree[lson[rt]] + tree[rson[rt]]; }
    inline void push_down(int rt, int m){
        if(!lazy[rt]) return;
        if(!lson[rt]) lson[rt] = ++tot;
        if(!rson[rt]) rson[rt] = ++tot;
        lazy[lson[rt]] += lazy[rt], lazy[rson[rt]] += lazy[rt];
        tree[lson[rt]] += lazy[rt] * (m - (m >> 1));
        tree[rson[rt]] += lazy[rt] * (m >> 1);
        lazy[rt] = 0;
    }

    static void update_part(int &rt ,int l, int r, int L, int R, int val){
        if(!rt) rt = ++tot;
        if(l >= L && r <= R){
            lazy[rt] += val;
            tree[rt] += val * (r - l + 1);
            return;
        }
        push_down(rt, r - l + 1);
        int mid = l + r >> 1;
        if(mid >= L) update_part(lson[rt], l, mid, L, R, val);
        if(mid < R) update_part(rson[rt], mid + 1, r, L, R, val);
        push_up(rt);
    }

    static void update_point(int &rt, int l, int r, int pos, int val){
        if(!rt) rt = ++tot;
        if(l == r){
            tree[rt] += val;
            return;
        }
        int mid = l + r >> 1;
        if(mid >= pos) update_point(lson[rt], l, mid, pos, val);
        else update_point(rson[rt], mid + 1, r, pos, val);
        push_up(rt);
    }

    static int query(int rt, int l, int r, int L, int R){
        if(!rt) return 0;
        if(l >= L && r <= R) return tree[rt];
        push_down(rt, r - l + 1);
        int mid = l + r >> 1, ans = 0;
        if(mid >= L) ans += query(lson[rt], l, mid, L, R);
        if(mid < R) ans += query(rson[rt], mid + 1, r, L, R);
        return ans;
    }
}//DynamicSegmentTree
```

6. 可持久化权值线段树 (主席树)

0. 模板题/模板封装

给定排列 $p_1, p_2, p_3, \ldots, p_n$, 定义 A_i 表示在 p_i 左侧并比 p_i 小的数字个数, B_i 表示在 p_i 右侧并比 p_i 小的数字个数, $C_i = \min(A_i, B_i)$。现在给定多个操作 (l, r)，求每个操作，交换 (p_i, p_j) 后的 $\sum C_i$。

首先考虑如何处理初始时的 C_i 值，观察到以下性质:

- 对于 A_i 值的求解过程类似求逆序对的思想，可以直接上树状数组维护，$O(n \log n)$ 求得全部的 A_i
- 由于是排列，$B_i = p_i - 1 - A_i$ 可以 $O(1)$ 求得
- 那么 $C_i = \min(A_i, B_i)$ 也是 $O(1)$ 得到的

由于每个询问相互独立，那么考虑交换 (p_l, p_r) 操作对 C_i 的影响:

- 对于 $[1,l),(r,n]$ 范围的数字，C_i 值一定不影响。因为交换操作均在单侧进行
- 对于 p_l，交换到 r 位置后，$A_l \to A_l +\ \ [l,r]\ \ p_l$ ，B_l' 仍然可以直接求

  对于 p_r，交换到 l 位置后，$A_r \to A_r -\ \ [l,r]\ \ p_r$ ，B_r' 仍然可以直接求

  如果我们在线询问 (主席树维护)，那么对于 p_l, p_r，实际上可以直接两个 $O(logn)$ 重新求。
- 那么重点是对于 $[l+1, r-1]$ 区间内的数字的 C_i 值变化，如何维护？
- 对于 $p_l \le p_i \le p_r$，如果 $A_i \le B_i$，则交换后 $A_i - 1, B_i + 1$，从而 $C_i - 1$

  对于 $p_l \ge p_i \ge p_r$，如果 $A_i \ge B_i$，则交换后 $A_i + 1, B_i - 1$，从而 $C_i - 1$
- 对于 $p_l \le p_i \le p_r$，如果 $A_i - 1 \ge B_i + 1, A_i \ge B_i$，则交换后 $C_i + 1$

  对于 $p_l \ge p_i \ge p_r$，如果 $A_i - 1 \le B_i + 1, A_i \le B_i$，则交换后 $C_i + 1$

那么对于以上四种情况，我们可以分别用四棵主席树进行维护。同时，对于 p_l, p_r 的贡献计算还需要支持区间 $< K$ 的数字个数查询，因此共需五棵主席树进行维护，复杂度 $O(m \times 4 \log n)$。

```
namespace PresidentTree{
    int root[N], sum[N << 5][5], lc[N << 5], rc[N << 5], cnt;
    #define ls l, mid
    #define rs mid + 1, r

    void update(int &rt, int pre, int l, int r, int x, bset5 inc){
        rt = ++cnt, lc[rt] = lc[pre], rc[rt] = rc[pre];
        for(int i = 0; i <= 5; i++) sum[rt][i] = sum[pre][i] + (inc[i] ? 1 : 0);
        if(l == r) return;
        int mid = l + r >> 1;
        if(x <= mid) update(lc[rt], lc[rt], l, mid, x, inc);
        else update(rc[rt], rc[rt], mid + 1, r, x, inc);
    }

    int query(int st, int ed, int l, int r, int L, int R, int id){
        if(l == L && r == R) return sum[ed][id] - sum[st][id];
        int mid = l + r >> 1;
        if(mid >= R) return query(lc[st], lc[ed], l, mid, L, R, id);
        else if(mid >= L) return query(lc[st], lc[ed], l, mid, L, mid, id) + query(rc[st], rc[ed], mid + 1, r, mid + 1, R, id);
        else return query(rc[st], rc[ed], mid + 1, r, L, R, id);
    }

}
```

1. 主席树维护静态区间第 K 大

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

const int N = 2e5 + 10;
int root[N], tot;
int lc[N << 5], rc[N << 5], sum[N << 5];
int a[N], b[N], n, m;

namespace cmt{
    void update(int &rt, int pre, int l, int r, int pos, int v){
        rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
        if(l == r) return;
        int mid = l + r >> 1;
        if(pos <= mid) update(lc[rt], lc[pre], l, mid, pos, v);
        else update(rc[rt], rc[pre], mid + 1, r, pos, v);
    }

    int query(int ql, int qr, int l, int r, int k){
        if(l == r) return l;
        int mid = l + r >> 1, summ = sum[lc[qr]] - sum[lc[ql]];
        if(summ >= k) return query(lc[ql], rc[qr], l, mid, k);
        else return query(rc[ql], rc[qr], mid + 1, r, k - summ);
    }
}
```

```
signed main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        b[i] = a[i];
    }
    sort(b + 1, b + 1 + n);
    int n_1 = unique(b + 1, b + 1 + n) - (b + 1);
    for(int i = 1; i <= n; i++) cmt::update(root[i], root[i - 1], 1, n_1, lower_bound(b + 1, b + 1 + n_1, a[i]) - b, 1);
    for(int i = 1; i <= m; i++){
        int l, r, k; cin >> l >> r >> k;
        cout << b[cmt::query(root[l - 1], root[r], 1, n_1, k)] << endl;
    }
    return 0;
}
```

2. 主席树维护线段树区间修改 (标记永久化)

主席树可以维护线段树的区间修改，但是要求线段树动态开点。

在区间更新的时候，对每个点打永久化标记，对于跨越区间的点需要直接修改。查询时找到对应的完全覆盖区间加上这个区间的标记 *(区间长度)

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;

const int N = 1e5 + 10;
int root[N], lc[N << 6], rc[N << 6], tot = 0, cur = 0;
ll sum[N << 6], lazy[N << 6];
int n, m;

void build(int &rt, int l, int r){
    rt = ++tot, lazy[rt] = 0;
    if(l == r){
        scanf("%lld", &sum[rt]);
        return;
    }
    int mid = l + r >> 1;
    build(lc[rt], l, mid);
    build(rc[rt], mid + 1, r);
    sum[rt] = sum[lc[rt]] + sum[rc[rt]];
}

void update(int &rt, int pre, int l, int r, int L, int R, int c){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], lazy[rt] = lazy[pre], sum[rt] = sum[pre] + 1ll * (min(r, R) - max(l,
    L) + 1) * c;
    if(L >= l && R <= r){
        lazy[rt] += c;
        return;
    }
    int mid = L + R >> 1;
    if(l <= mid) update(lc[rt], lc[pre], l, r, L, mid, c);
    if(r > mid) update(rc[rt], rc[pre], l, r, mid + 1, R, c);
}

ll query(int rt, int L, int R, int l, int r){
    if(L >= l && R <= r) return sum[rt];
    int mid = L + R >> 1;
    ll ans = lazy[rt] * (min(r, R) - max(l, L) + 1);
    if(l <= mid) ans += query(lc[rt], L, mid, l, r);
    if(r > mid) ans += query(rc[rt], mid + 1, R, l, r);
    return ans;
}

signed main(){
    while(scanf("%d%d", &n, &m) != EOF){
        char op[10];
        cur = tot = 0;
        build(root[0], 1, n);
```

```
        while(m--){
            scanf("%s", op);
            if(op[0] == 'C'){
                int l, r, c;
                scanf("%d%d%d", &l, &r, &c);
                cur++;
                update(root[cur], root[cur - 1], l, r, 1, n, c);
            }
            else if(op[0] == 'Q'){
                int l, r;
                scanf("%d%d", &l, &r);
                printf("%lld\n", query(root[cur], 1, n, l, r));
            }
            else if(op[0] == 'H'){
                int l, r, h;
                scanf("%d%d%d", &l, &r, &h);
                printf("%lld\n", query(root[h], 1, n, l, r));
            }
            else if(op[0] == 'B'){
                scanf("%d", &cur);
            }
        }
    }
    return 0;
}
```

3. 求区间内不同的数字个数/求区间大于 K 的数字有多少

求区间内不同数字个数的问题可以转化为求区间内小于等于 K 的数字个数:

对于每个数字记录下一个最近的相同数字下标 $nxt[i]$，那么查询区间 $[L,R]$ 内不同数字的个数实际上就是在查询区间内 $nxt[i] > R$ 的个数 (下一个相同的数字位于区间之外)。那么现在不难发现对于给定区间 $[l,r]$，如果 $nxt[i] > r$，$i \in [l,r]$，那么表示与 i 相同数字点位于区间之外。那么求不同数字个数问题便转化为给定求所有满足 $l <= i <= r$，$nxt[i] > r$ 的个数。

那么我们只需要处理出 nxt 数组，然后用主席树对每个节点维护权值数组，然后区间查询数目即可。

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

const int N = 1e6 + 10;
int a[N], head[N], nxt[N];
int root[N], sum[N << 5], lc[N << 5], rc[N << 5], cnt;

inline int read(){
    int f = 1, x = 0; char s = getchar();
    while(s < '0'||s > '9'){ if(s =='-') f = -1; s = getchar(); }
    while(s >= '0' && s <= '9'){ x = x * 10 + s - '0'; s = getchar();}
    return x *= f;
}

void build(int &rt, int l, int r){
    rt = ++cnt;
    if(l == r) return;
    int mid = l + r >> 1;
    build(lc[rt], l, mid);
    build(rc[rt], mid + 1, r);
}

void update(int &rt, int pre, int l, int r, int x){
    rt = ++cnt, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(x <= mid) update(lc[rt], lc[pre], l, mid, x);
    else update(rc[rt], rc[pre], mid + 1, r, x);
}

int query(int L, int R, int l, int r, int k){
    if(l == r) return sum[R] - sum[L];
    int mid = l + r >> 1, ans = 0;
    if(k <= mid) ans += query(lc[L], lc[R], l, mid, k) + sum[rc[R]] - sum[rc[L]];
    else ans += query(rc[L], rc[R], mid + 1, r, k);
```

```cpp
    return ans;
}

signed main(){
    int n = 0;
    n = read();
    for(int i = 1; i <= n; i++){
        a[i] = read();  //cin >> a[i];
        if(head[a[i]]) nxt[head[a[i]]] = i;
        head[a[i]] = i;
    }
    for(int i = 1; i <= n; i++)
        if(!nxt[i]) nxt[i] = n + 1;

    build(root[0], 1, n + 1);
    for(int i = 1; i <= n; i++) update(root[i], root[i - 1], 1, n + 1, nxt[i]);

    int m = read();
    while(m--){
        int l = read(), r = read();
        printf("%d\n", query(root[l - 1], root[r], 1, n + 1, r + 1));
    }
    return 0;
}
```

4. 求区间小于等于 K 的数字个数 (二分查询)

建立权值数组，对每个节点维护一颗主席树。查询为单点查询，查询数字对应的权值数组的个数。然后对于每个询问，我们在区间内二分枚举所有可能的数字 k，然后查询区间第 k 小。反复查询求得一个满足 $k \leq h$ 的最大 k。那么这个 k 就是我们想要得到的答案。

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;

const int N = 1e5 + 10;

ll root[N], sum[N << 5], lc[N << 5], rc[N << 5], tot = 0;
ll a[N], b[N];

inline void update(ll &rt, ll pre, ll l, ll r, ll k){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    ll mid = l + r >> 1;
    if(l == r) return;
    if(k <= mid) update(lc[rt], lc[pre], l, mid, k);
    else update(rc[rt], rc[pre], mid + 1, r, k);
}

ll query(ll u, ll v, ll L, ll R, ll k){
    if(L == R) return L;
    ll mid = L + R >> 1;
    ll res = sum[lc[v]] - sum[lc[u]];
    if(res >= k) return query(lc[u], lc[v], L, mid, k);
    else return query(rc[u], rc[v], mid + 1, R, k - res);
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int t = 0, T = 0; cin >> t;
    while(t--){
        ll n, m; cin >> n >> m;
        for(int i = 1; i <= n; i++){
            cin >> a[i]; b[i] = a[i];
        }
        sort(b + 1, b + 1 + n);
        ll sz = unique(b + 1, b + 1 + n) - b - 1;
        for(int i = 1; i <= n; i++){
            ll x = lower_bound(b + 1, b + 1 + sz, a[i]) - b;
            update(root[i], root[i - 1], 1, sz, x);
        }
        cout << "Case " << ++T << ":" << endl;
        while(m--){
```

```cpp
            ll u, v, k; cin >> u >> v >> k;
            u++, v++;
            ll ans = 0, L = 0, R = v - u + 1;
            while(L < R){
                ll mid = (L + R + 1) >> 1;
                ll t = query(root[u - 1], root[v], 1, sz, mid);
                if(b[t] <= k) L = mid;
                else R = mid - 1;
            }
            cout << L << endl;
        }
    }
    return 0;
}
```

5. 求区间 Mex

给定 n 长度的数组，$\{a_1, a_2, ..., a_n\}$，以及 m 次询问，每次给出一个数对 (l, r) 表示区间起点终点，要求对于给定的询问，回答在该区间内最小未出现的数字。

建立权值数组，对于每个点建立一棵主席树，维护权值最后一次出现的位置，那么对于查询 $[l, r]$ 就是查找第 r 棵树上出现位置小于 l 的权值，那么只需要维护最后一次出现位置的最小值即可。

主席树解决该问题属于在线算法。这种题目可以用莫队强制离线处理。

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

const int N = 2e5 + 10;

int tot, root[N], tree[N << 5], lc[N << 5], rc[N << 5];

void update(int &rt, int pre, int l, int r, int x, int val){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre];
    if(l == r){
        tree[rt] = val;
        return;
    }
    int mid = l + r >> 1;
    if(x <= mid) update(lc[rt], lc[pre], l, mid, x, val);
    else update(rc[rt], rc[pre], mid + 1, r, x, val);
    tree[rt] = min(tree[lc[rt]], tree[rc[rt]]);
}

int query(int rt, int ql, int l, int r){
    if(l == r) return l;
    int mid = l + r >> 1;
    if(tree[lc[rt]] < ql) return query(lc[rt], ql, l, mid);
    else return query(rc[rt], ql, mid + 1, r);
}


signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int n, m; cin >> n >> m;
    for(int i = 1, x; i <= n; i++){
        cin >> x; x++;
        if(x > n) root[i] = root[i - 1];
        else update(root[i], root[i - 1], 1, n + 1, x, i);
    }
    while(m--){
        int l, r; cin >> l >> r;
        cout << query(root[r], l, 1, n + 1) - 1 << endl;
    }
    return 0;
}
```

6. 求区间内出现次数大于 >=k 次的最前数

对于给定的序列，输出待查询区间内出现次数严格大于区间长度一半的数字。

思路：考虑对查询过程进行剪枝，排除非法子树，向合法子树搜索。

首先考虑非法状态：因为对于主席树上任意一个节点，其代表的意义是管辖区间内数字的个数。因此对于主席树上某个节点，如果其代表区间数字的数目比区间长度的一半 (也就是 $\frac{r-l+1}{2}$) 要小，那么子区间不回再出现满足该条件的数，在这种情况下可以直接返回 0。

剩下的部分就是查询的板子。非法状态实际上就是在对查询过程进行剪枝。

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

const int N = 5e5 + 10;

int a[N], b[N];
int tot, root[N << 5], sum[N << 5], lc[N << 5], rc[N << 5];

void update(int &rt, int pre, int l, int r, int v){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(v <= mid) update(lc[rt], lc[pre], l, mid, v);
    else update(rc[rt], rc[pre], mid + 1, r, v);
}

int query(int L, int R, int l, int r, int k){
    if(sum[R] - sum[L] <= k) return 0;
    if(l == r) return l;
    int now = sum[lc[R]] - sum[lc[L]], mid = l + r >> 1;
    if(now > k) return query(lc[L], lc[R], l, mid, k);
    else return query(rc[L], rc[R], mid + 1, r, k);
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int n, q; cin >> n >> q;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        b[i] = a[i];
    }
    sort(b + 1, b + n + 1);
    int m = unique(b + 1, b + n + 1) - b - 1;
    for(int i = 1; i <= n; i++){
        int x = lower_bound(b + 1, b + m + 1, a[i]) - b;
        update(root[i], root[i - 1], 1, m, x);
    }
    while(q--){
        int l, r; cin >> l >> r;
        int k = (r - l + 1) >> 1;
        cout << b[query(root[l - 1], root[r], 1, m, k)] << endl;
    }
    return 0;
}
```

7. 主席树 + 树上路径

给定一棵 n 个节点的树，每个点有一个权值。有 m 个询问，每次给你 u, v, k 你需要回答 u xor last 和 v 这两个节点间第 k 小的点权。

动态查询树上区间点权第 k 小，且查询之间具有关系，因此考虑建立主席树维护区间信息。

首先回顾主席树维护线性区间第 k 大/小时我们的处理思路：

对于全局区间第 k 小时，我们建立全局权值线段树，维护的区间和表示某点子树中点的个数。那么我们在寻找区间第 k 小时，只需要左右二分寻找即可。而对于某个区间的第 k 小，一个朴素的方式便是我们每次都建立一颗线段树，但显然这样是不明智的算法。那么我们是如何查询这个区间第 k 小的呢？

对于这个维护的过程我们很容易联想到前缀和的概念，我们可以先离线建树，对于每个点建立一棵主席树，维护 $[1, i]$ 区间，那么对于区间查询 $[l, r]$ 时，我们只需要查询到区间 $[1, l-1]$ 和 $[1, r]$ 即可取得区间的信息，实现区间查询。

然后分析样例，作出样例所示的树 (假设以 1 为根节点)：

那么我们可以发现，对于树上区间查询，我们也可以利用类似于线性区间查询的思路进行解决，但是由于树的结构限制，我们把线性区间

的前缀和改为树上前缀和的形式：$query(u,\ v) = sum[u] + sum[v] - sum[lca(u,\ v)] - sum[fa[lca(u,\ v)]]$ 下面我们来说明这个式子：

如上，从根节点到 5 号节点的路径 + 从根节点到 7 号节点的路径重复了两次，那么我们要减去重叠的信息：对于根节点到交点父节点的信息均重复两次，到交点的信息重复一次 (因为交点在链上，需要保留一次信息)，因此前缀和形式便是 $sum[u] + sum[v] - sum[lca(u,\ v)] - sum[fa[lca(u,\ v)]]$。

```cpp
#include <bits/stdc++.h>
#define id(x) (lower_bound(b + 1, b + le + 1, a[x]) - b)
#define rid(x) (b[x])
using namespace std;

const int N = 1e5 + 10;

int n, m, le, ans = 0, lastans = 0;
int a[N], b[N], f[N][19], dep[N];
vector<int> g[N];

struct node{ int sum, lc, rc; }tree[N << 5];
int root[N], cnt = 0;


void build(node &rt, int l, int r){
    rt.sum = 0;
    if(l == r) return;
    int mid = l + r >> 1;
    build(tree[rt.lc = ++cnt], l, mid);
    build(tree[rt.rc = ++cnt], mid + 1, r);
}

inline void update(node pre, node &rt, int l, int r, int p){
    rt.sum = pre.sum + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(p <= mid) update(tree[pre.lc], tree[rt.lc = ++cnt], l, mid, p), rt.rc= pre.rc;
    else update(tree[pre.rc], tree[rt.rc = ++cnt], mid + 1, r, p), rt.lc = pre.lc;
}

inline int query(node u, node v, node lca, node lca_fa, int l, int r, int k){
    if(l == r) return l;
    int sum = tree[u.lc].sum + tree[v.lc].sum - tree[lca.lc].sum - tree[lca_fa.lc].sum;
    int mid = l + r >> 1;
    if(sum >= k) return query(tree[u.lc], tree[v.lc], tree[lca.lc], tree[lca_fa.lc], l, mid, k);
    return query(tree[u.rc], tree[v.rc], tree[lca.rc], tree[lca_fa.rc], mid + 1, r, k - sum);
}

inline void dfs(int u, int fa){
    update(tree[root[fa]], tree[root[u] = ++cnt], 1, le, id(u));
    f[u][0] = fa;
    dep[u] = dep[fa] + 1;
    for(register int i = 1; i <= 18; i++) f[u][i] = f[f[u][i - 1]][i - 1];
    for(auto v : g[u]){
        if(v == fa) continue;
        dfs(v, u);
    }
}

inline int lca(int u, int v){
    if(dep[u] < dep[v]) swap(u, v);
    for(register int i = 18; i >= 0; i--)
        if(dep[f[u][i]] >= dep[v]) u = f[u][i];
    if(u == v) return u;
    for(register int i = 18; i >= 0; i--)
        if(f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
    return f[u][0];
}

inline int querypath(int u, int v, int k){
    int lcaa = lca(u, v);
    return rid(query(tree[root[u]], tree[root[v]], tree[root[lcaa]], tree[root[f[lcaa][0]]], 1, le, k));
}
```

```cpp

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    //freopen("stdin.in", "r", stdin);
    //freopen("stdout.out", "w", stdout);
    cin >> n >> m;
    for(register int i = 1; i <= n; i++){ cin >> a[i]; b[i] = a[i]; }
    for(register int i = 1, u, v; i < n; i++){
        cin >> u >> v;
        g[u].push_back(v), g[v].push_back(u);
    }
    sort(b + 1, b + 1 + n);
    le = unique(b + 1, b + n + 1) - (b + 1);
    build(tree[root[0] = ++cnt], 1, le);
    dfs(1, 0);
    while(m--){
        int u, v, k; cin >> u >> v >> k;
        ans = querypath(u ^ lastans, v, k);
        cout << ans << endl;
        lastans = ans;
    }
    return 0;
}
```

7. 树套树

树状数组套主席树

动态区间第 k 大

```cpp
typedef vector<int> VI;
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20 * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int add(int tt, int l, int r, int x, int d) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + d, 0, 0);
        return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson, x, d));
    }
    int ls_sum(const VI& rt) {
        int ret = 0;
        FOR (i, 0, rt.size())
            ret += tr[tr[rt[i]].ls].w;
        return ret;
    }
    inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].ls; }); }
    inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].rs; }); }
    int query(VI& p, VI& q, int l, int r, int k) {
        if (l == r) return l;
        int w = ls_sum(q) - ls_sum(p);
        if (k <= w) {
            ls(p); ls(q);
            return query(p, q, lson, k);
        }
        else {
            rs(p); rs(q);
            return query(p, q, rson, k - w);
        }
    }
} tree;
```

```cpp
struct BIT {
    int root[maxn];
    void init() { memset(root, 0, sizeof root); }
    inline int lowbit(int x) { return x & -x; }
    void update(int p, int x, int d) {
        for (int i = p; i <= m; i += lowbit(i))
            root[i] = tree.add(root[i], 1, m, x, d);
    }
    int query(int l, int r, int k) {
        VI p, q;
        for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(root[i]);
        for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[i]);
        return tree.query(p, q, 1, m, k);
    }
} bit;

void init() {
    m = 10000;
    tree.sz = 1;
    bit.init();
    FOR (i, 1, m + 1)
        bit.update(i, a[i], 1);
}
```

8.K-D Tree

在一个初始值全为 0 的 $n \times n$ 的二维矩阵上，进行 q 次操作，每次操作为以下两种之一：

1. ***1 x y A***：将坐标 (x, y) 上的数加上 A。
2. ***2 x1 y1 x2 y2***：输出以 (x_1, y_1) 为左下角，(x_2, y_2) 为右上角的矩形内（包括矩形边界）的数字和。

强制在线。内存限制 ***20M***。保证答案及所有过程量在 ***int*** 范围内。

构建 2-D Tree，支持两种操作：添加一个 2 维点；查询矩形区域内的所有点的权值和。可以使用 **带重构**的 k-D Tree 实现。

在查询矩形区域内的所有点的权值和时，仍然需要记录子树内每一维度上的坐标的最大值和最小值。如果当前子树对应的矩形与所求矩形没有交点，则不继续搜索其子树；如果当前子树对应的矩形完全包含在所求矩形内，返回当前子树内所有点的权值和；否则，判断当前点是否在所求矩形内，更新答案并递归在左右子树中查找答案。

已经证明，如果在 $2-D$ 树上进行矩阵查询操作，已经被完全覆盖的子树不会继续查询，则单次查询时间复杂度是最优 $O(\log n)$，最坏 $O(\sqrt{n})$ 的。将结论扩展到 k 维的情况，则最坏时间复杂度是 $O(n^{1-\frac{1}{k}})$ 的。

```cpp
const int maxn = 200010;
int n, op, xl, xr, yl, yr, lstans;

struct node{ int x, y, v; } s[maxn];

bool cmp1(int a, int b) { return s[a].x < s[b].x; }

bool cmp2(int a, int b) { return s[a].y < s[b].y; }

double a = 0.725;
int rt, cur, d[maxn], lc[maxn], rc[maxn], L[maxn], R[maxn], D[maxn], U[maxn],
    siz[maxn], sum[maxn];
int g[maxn], t;

void print(int x){
    if (!x) return;
    print(lc[x]);
    g[++t] = x;
    print(rc[x]);
}

void maintain(int x){
    siz[x] = siz[lc[x]] + siz[rc[x]] + 1;
    sum[x] = sum[lc[x]] + sum[rc[x]] + s[x].v;
    L[x] = R[x] = s[x].x;
    D[x] = U[x] = s[x].y;
    if (lc[x])
        L[x] = min(L[x], L[lc[x]]), R[x] = max(R[x], R[lc[x]]),
```

```
        D[x] = min(D[x], D[lc[x]]), U[x] = max(U[x], U[lc[x]]);
    if (rc[x])
        L[x] = min(L[x], L[rc[x]]), R[x] = max(R[x], R[rc[x]]),
        D[x] = min(D[x], D[rc[x]]), U[x] = max(U[x], U[rc[x]]);
}

int build(int l, int r){
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
    for (int i = l; i <= r; i++)
        av1 += s[g[i]].x, av2 += s[g[i]].y;
    av1 /= (r - l + 1);
    av2 /= (r - l + 1);
    for (int i = l; i <= r; i++)
        va1 += (av1 - s[g[i]].x) * (av1 - s[g[i]].x), va2 += (av2 - s[g[i]].y) * (av2 - s[g[i]].y);
    if (va1 > va2) nth_element(g + l, g + mid, g + r + 1, cmp1), d[g[mid]] = 1;
    else nth_element(g + l, g + mid, g + r + 1, cmp2), d[g[mid]] = 2;
    lc[g[mid]] = build(l, mid - 1);
    rc[g[mid]] = build(mid + 1, r);
    maintain(g[mid]);
    return g[mid];
}

void rebuild(int &x){
    t = 0;
    print(x);
    x = build(1, t);
}

bool bad(int x) { return a * siz[x] <= (double)max(siz[lc[x]], siz[rc[x]]); }

void insert(int &x, int v){
    if (!x){
        x = v;
        maintain(x);
        return;
    }
    if (d[x] == 1){
        if (s[v].x <= s[x].x) insert(lc[x], v);
        else insert(rc[x], v);
    }
    else{
        if (s[v].y <= s[x].y) insert(lc[x], v);
        else insert(rc[x], v);
    }
    maintain(x);
    if (bad(x)) rebuild(x);
}

int query(int x){
    if (!x || xr < L[x] || xl > R[x] || yr < D[x] || yl > U[x]) return 0;
    if (xl <= L[x] && R[x] <= xr && yl <= D[x] && U[x] <= yr) return sum[x];
    int ret = 0;
    if (xl <= s[x].x && s[x].x <= xr && yl <= s[x].y && s[x].y <= yr) ret += s[x].v;
    return query(lc[x]) + query(rc[x]) + ret;
}

int main(){
    scanf("%d", &n);
    while (~scanf("%d", &op)){
        if (op == 1){
            cur++, scanf("%d%d%d", &s[cur].x, &s[cur].y, &s[cur].v);
            s[cur].x ^= lstans;
            s[cur].y ^= lstans;
            s[cur].v ^= lstans;
            insert(rt, cur);
        }
        if (op == 2){
            scanf("%d%d%d%d", &xl, &yl, &xr, &yr);
            xl ^= lstans;
```

```
                yl ^= lstans;
                xr ^= lstans;
                yr ^= lstans;
                printf("%d\n", lstans = query(rt));
            }
            if (op == 3) return 0;
        }
}
/* Test Case
//INPUT
4
1 2 3 3
2 1 1 3 3
1 1 1 1
2 1 1 0 7
3
//OUTPUT
3
5
*/
```

9.Trie(字典树)

A.0-1 Trie

01 字典树主要用于解决求异或最值的问题

```
namespace Trie01{
    int tot, nxt[20*N][2];
    inline void clear() { tot = nxt[0][0] = nxt[0][1] = 0; }
    void insert(int x) {
        int rt = 0;
        for (int i = 20; i >= 0; --i) {
            int cur = (x >> i & 1);
            if (!nxt[rt][cur]) {
                nxt[rt][cur] = ++tot;
                nxt[tot][0] = nxt[tot][1] = 0;
            }
            rt = nxt[rt][cur];
        }
    }

    int query_min(int x) {
        int rt = 0, ans = 0;
        for (int i = 20; i >= 0; -- i) {
            int cur = (x >> i & 1), need = cur;
            if (!nxt[rt][need]) ans = (ans << 1) | 1, need = !need;
            else ans = (ans << 1);
            rt = nxt[rt][need];
        }
        return ans;
    }

    int query_max(int x) {
        int rt = 0, ans = 0;
        for (int i = 20; i >= 0; -- i) {
            int cur = (x >> i & 1), need = (cur ^ 1);
            if (!nxt[rt][need]) ans = (ans << 1), need = !need;
            else ans = (ans << 1) | 1;
            rt = nxt[rt][need];
        }
        return ans;
    }
}
```

B.Normal Trie

给定 n 个模式串 $s_1, s_2, \dots, s_n$ 和 q 次询问，每次询问给定一个文本串 t_i，请回答 $s_1 \sim s_n$ 中有多少个字符串 s_j 满足 t_i 是 s_j 的**前缀**。保证 $1 \le T, n, q \le 10^5$，且输入字符串的总长度不超过 3×10^6。

```cpp
const int N = 3e6 + 10, MOD = 1e9 + 7;
const int DICT_SIZE = 65;
namespace Trie{
    int tot, nxt[N][DICT_SIZE], cnt[N];

    void clear(){
        for(int i = 0; i <= tot; i++){
            cnt[i] = 0;
            for(int j = 0; j <= 122; j++) nxt[i][j] = 0;
        }
        tot = 0;
    }

    int getnum(char x){
        if(x >= 'A' && x <= 'Z') return x - 'A';
        else if(x >= 'a' && x <= 'z') return x - 'a' + 26;
        else return x - '0' + 52;
    }

    void insert(char str[]){
        int p = 0, len = strlen(str);
        for(int i = 0; i < len; i++){
            int c = getnum(str[i]);
            if(!nxt[p][c]) nxt[p][c] = ++tot;
            p = nxt[p][c];
            cnt[p]++;
        }
    }

    int find(char str[]){
        int p = 0, len = strlen(str);
        for(int i = 0; i < len; i++){
            int c = getnum(str[i]);
            if(!nxt[p][c]) return 0;
            p = nxt[p][c];
        }
        return cnt[p];
    }
}

char s[N];

inline void solve(){
    Trie::clear();
    int n, q; cin >> n >> q;
    for(int i = 0; i < n; i++){
        cin >> s; Trie::insert(s);
    }
    for(int i = 0; i < q; i++){
        cin >> s; cout << Trie::find(s) << endl;
    }
}
/* TEST CASE
//INPUT
3
3 3
fusufusu
fusu
anguei
fusu
anguei
kkksc
5 2
fusu
Fusu
AFakeFusu
afakefusu
fusuisnotfake
Fusu
fusu
1 1
```

```
998244353
9
//OUTPUT
2
1
0
1
2
1
```

10. 笛卡尔树

给定一个 $1 \sim n$ 的排列 p，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。
2. 节点 i 的权值为 p_i，每个节点的权值满足小根堆的性质。

这棵树的每个结点有两个子树，分为左右子树，子树可以为空;

一个结点的左子树中的所有结点的第一个权值都小于其第一个权值（空子树也满足）;

一个结点的右子树中的所有结点的第一个权值都大于其第一个权值（空子树也满足）;

一个结点的两棵子树中的所有结点的第二个权值都大于其第二个权值（空子树也满足）。

```cpp
const int N = 1e7 + 7;
int n, a[N], stk[N], ls[N], rs[N];

void build(int n){
    for (int i = 1, pos = 0, top = 0; i <= n; ++i){ //这是按下标顺序插入元素的代码
        pos = top;
        while (pos && a[stk[pos]] > a[i]) pos--;
        if (pos) rs[stk[pos]] = i;
        if (pos < top) ls[i] = stk[pos + 1];
        stk[top = ++pos] = i;
    }
}

inline void solve(){
    int n = read();
    for(int i = 1; i <= n; i++) a[i] = read();
    build(n);
    long long L = 0, R = 0;
    for (int i = 1; i <= n; ++i)
        L ^= 1LL * i * (ls[i] + 1), R ^= 1LL * i * (rs[i] + 1);
    printf("%lld %lld", L, R);
}
/*TEST CASE
//INPUT
5
4 1 3 2 5
//OUTPUT
19 21
*/
```

11.Treap

FROM ECNU 板子库

原始 Treap

- 非旋 Treap
- v 小根堆
- lower 第一个大于等于的是第几个 (0-based)
- upper 第一个大于的是第几个 (0-based)
- split 左侧分割出 rk 个元素

```cpp
namespace treap {
    const int M = maxn * 17;
    extern struct P* const null;
    struct P {
        P *ls, *rs;
        int v, sz;
        unsigned rd;
        P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
        P(): sz(0) {}

        P* up() { sz = ls->sz + rs->sz + 1; return this; }
        int lower(int v) {
            if (this == null) return 0;
            return this->v >= v ? ls->lower(v) : rs->lower(v) + ls->sz + 1;
        }
        int upper(int v) {
            if (this == null) return 0;
            return this->v > v ? ls->upper(v) : rs->upper(v) + ls->sz + 1;
        }
    } *const null = new P, pool[M], *pit = pool;

    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->up(); }
        else { r->ls = merge(l, r->ls); return r->up(); }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o->up(); }
        else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o->up(); }
    }
}
```

- 持久化 Treap

```cpp
namespace treap {
    const int M = maxn * 17 * 12;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int v, sz;
        LL sum;
        P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz + rs->sz + 1),
                                                  sum(ls->sum + rs->sum + v) {}
        P() {}

        void* operator new(size_t _) { return pit++; }
        template<typename T>
        int rk(int v, T&& cmp) {
            if (this == null) return 0;
            return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v, cmp) + ls->sz + 1;
        }
        int lower(int v) { return rk(v, greater_equal<int>()); }
        int upper(int v) { return rk(v, greater<int>()); }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (rnd() % (l->sz + r->sz) < l->sz) return new P{l->ls, merge(l->rs, r), l->v};
        else return new P{merge(l, r->ls), r->rs, r->v};
    }
    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P{r, o->rs, o->v}; }
        else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P{o->ls, l, o->v}; }
    }
}
```

- 带 pushdown 的持久化 Treap
- 注意任何修改操作前一定要 FIX

```cpp
int now;
```

```
namespace Treap {
    const int M = 10000000;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int sz, time;
        LL cnt, sc, pos, add;
        bool rev;

        P* up() { sz = ls->sz + rs->sz + 1; sc = ls->sc + rs->sc + cnt; return this; } // MOD
        P* check() {
            if (time == now) return this;
            P* t = new(pit++) P; *t = *this; t->time = now; return t;
        };
        P* _do_rev() { rev ^= 1; add *= -1; pos *= -1; swap(ls, rs); return this; } // MOD
        P* _do_add(LL v) { add += v; pos += v; return this; } // MOD
        P* do_rev() { if (this == null) return this; return check()->_do_rev(); } // FIX & MOD
        P* do_add(LL v) { if (this == null) return this; return check()->_do_add(v); } // FIX & MOD
        P* _down() { // MOD
            if (rev) { ls = ls->do_rev(); rs = rs->do_rev(); rev = 0; }
            if (add) { ls = ls->do_add(add); rs = rs->do_add(add); add = 0; }
            return this;
        }
        P* down() { return check()->_down(); } // FIX & MOD
        void _split(LL p, P*& l, P*& r) { // MOD
            if (pos >= p) { ls->split(p, l, r); ls = r; r = up(); }
            else          { rs->split(p, l, r); rs = l; l = up(); }
        }
        void split(LL p, P*& l, P*& r) { // FIX & MOD
            if (this == null) l = r = null;
            else down()->_split(p, l, r);
        }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* a, P* b) {
        if (a == null) return b; if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz) { a = a->down(); a->rs = merge(a->rs, b); return a->up(); }
        else                                  { b = b->down(); b->ls = merge(a, b->ls); return b->up(); }
    }
}
```

Treap-序列

- 区间 ADD，SUM

```
namespace treap {
    const int M = 8E5 + 100;
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, val, add, sum;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), val(v), add(0), sum(v) {}
        P(): sz(0), val(0), add(0), sum(0) {}

        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val + add * sz;
            return this;
        }
        void upd(int v) {
            if (this == null) return;
            add += v;
            sum += sz * v;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                val += add;
                add = 0;
            }
            return this;
```

```
        }

        P* select(int rk) {
            if (rk == ls->sz + 1) return this;
            return ls->sz >= rk ? ls->select(rk) : rs->select(rk - ls->sz - 1);
        }
    } pool[M], *pit = pool, *const null = new P, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b->up();
        if (b == null) return a->up();
        if (rand() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= rk) {
            split(o->ls, rk, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, rk - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    inline void insert(int k, int v) {
        P *l, *r;
        split(rt, k - 1, l, r);
        rt = merge(merge(l, new (pit++) P(v)), r);
    }

    inline void erase(int k) {
        P *l, *r, *_, *t;
        split(rt, k - 1, l, t);
        split(t, 1, _, r);
        rt = merge(l, r);
    }

    P* build(int l, int r, int* a) {
        if (l > r) return null;
        if (l == r) return new(pit++) P(a[l]);
        int m = (l + r) / 2;
        return (new(pit++) P(a[m], build(l, m - 1, a), build(m + 1, r, a)))->up();
    }
};
```

- 区间 REVERSE，ADD，MIN

```
namespace treap {
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, v, add, m;
        bool flip;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), v(v), add(0), m(v), flip(0) {}
        P(): sz(0), v(INF), m(INF) {}

        void upd(int v) {
            if (this == null) return;
            add += v; m += v;
        }
        void rev() {
            if (this == null) return;
            swap(ls, rs);
            flip ^= 1;
```

```
        }
        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            m = min(min(ls->m, rs->m), v) + add;
            return this;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                v += add;
                add = 0;
            }
            if (flip) {
                ls->rev(); rs->rev();
                flip = 0;
            }
            return this;
        }

        P* select(int k) {
            if (ls->sz + 1 == k) return this;
            if (ls->sz >= k) return ls->select(k);
            return rs->select(k - ls->sz - 1);
        }

    } pool[M], *const null = new P, *pit = pool, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rnd() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int k, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= k) {
            split(o->ls, k, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, k - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    P* build(int l, int r, int* v) {
        if (l > r) return null;
        int m = (l + r) >> 1;
        return (new (pit++) P(v[m], build(l, m - 1, v), build(m + 1, r, v)))->up();
    }

    void go(int x, int y, void f(P*&)) {
        P *l, *m, *r;
        split(rt, y, l, r);
        split(l, x - 1, l, m);
        f(m);
        rt = merge(merge(l, m), r);
    }
}
using namespace treap;
int a[maxn], n, x, y, Q, v, k, d;
char s[100];

int main() {
```

```cpp
    cin >> n;
    FOR (i, 1, n + 1) scanf("%d", &a[i]);
    rt = build(1, n, a);
    cin >> Q;
    while (Q--) {
        scanf("%s", s);
        if (s[0] == 'A') {
            scanf("%d%d%d", &x, &y, &v);
            go(x, y, [](P*& o){ o->upd(v); });
        } else if (s[0] == 'R' && s[3] == 'E') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o){ o->rev(); });
        } else if (s[0] == 'R' && s[3] == 'O') {
            scanf("%d%d%d", &x, &y, &d);
            d %= y - x + 1;
            go(x, y, [](P*& o){
                P *l, *r;
                split(o, o->sz - d, l, r);
                o = merge(r, l);
            });
        } else if (s[0] == 'I') {
            scanf("%d%d", &k, &v);
            go(k + 1, k, [](P*& o){ o = new (pit++) P(v); });
        } else if (s[0] == 'D') {
            scanf("%d", &k);
            go(k, k, [](P*& o){ o = null; });
        } else if (s[0] == 'M') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o) {
                printf("%d\n", o->m);
            });
        }
    }
}

```

- 持久化

```cpp
namespace treap {
    struct P;
    extern P*const null;
    P* N(P* ls, P* rs, LL v, bool fill);
    struct P {
        P *const ls, *const rs;
        const int sz, v;
        const LL sum;
        bool fill;
        int cnt;

        void split(int k, P*& l, P*& r) {
            if (this == null) { l = r = null; return; }
            if (ls->sz >= k) {
                ls->split(k, l, r);
                r = N(r, rs, v, fill);
            } else {
                rs->split(k - ls->sz - fill, l, r);
                l = N(ls, l, v, fill);
            }
        }


    } *const null = new P{0, 0, 0, 0, 0, 0, 1};

    P* N(P* ls, P* rs, LL v, bool fill) {
        ls->cnt++; rs->cnt++;
        return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum + rs->sum + v, fill, 1};
    }

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz)
```

```
            return N(a->ls, merge(a->rs, b), a->v, a->fill);
        else
            return N(merge(a, b->ls), b->rs, b->v, b->fill);
    }

    void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
        o->split(y, l, r);
        l->split(x - 1, l, m);
    }
}
```

12. 莫队

扩展顺序

```
while (l > q.l) mv(--l, 1);
while (r < q.r) mv(r++, 1);
while (l < q.l) mv(l++, -1);
while (r > q.r) mv(--r, -1);
```

13.CDQ 分治

```
const int maxn = 2E5 + 100;
struct P {
    int x, y;
    int* f;
    bool d1, d2;
} a[maxn], b[maxn], c[maxn];
int f[maxn];

void go2(int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go2(l, m); go2(m, r);
    for(int i = l; i < m; i++) b[i].d2 = 0;
    for(int i = m; i < r; i++) b[i].d2 = 1;
    merge(b + l, b + m, b + m, b + r, c + l, [](const P& a, const P& b)->bool {
            if (a.y != b.y) return a.y < b.y;
            return a.d2 > b.d2;
        });
    int mx = -1;
    for(int i = l; i < r; i++) {
        if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
        if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
    }
    for(int i = l; i < r; i++)  b[i] = c[i];
}

void go1(int l, int r) { // [l, r)
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go1(l, m);
    for(int i = l; i < m; i++) a[i].d1 = 0;
    for(int i = m; i < r; i++) a[i].d1 = 1;
    copy(a + l, a + r, b + l);
    sort(b + l, b + r, [](const P& a, const P& b)->bool {
            if (a.x != b.x) return a.x < b.x;
            return a.d1 > b.d1;
        });
    go2(l, r);
    go1(m, r);
}

```

14. 珂朵莉树/老司机树 (区间推平)

```cpp
struct node{
    int l, r;
    mutable int val;
    node (int lpos): l(lpos) {}
    node (int lpos, int rpos, int vall): l(lpos), r(rpos), val(vall) {}
    bool operator< (const node &a) const { return l < a.l; }
};

set<node> s;
using sit = set<node>::iterator;

sit split(int pos){
    sit it = s.lower_bound(node(pos));
    if(it != s.end() && it -> l == pos) return it;
    --it;
    int l = it -> l, r = it -> r, val = it -> val;
    s.erase(it);
    s.insert(node(l, pos - 1, val));
    return s.insert(node(pos, r, val)).first;
}

void assign(int l, int r, int val){
    sit itr = split(r + 1), itl = split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, val));
}

void add(int l, int r, int val){
    sit itr = split(r + 1), itl = split(l);
    //for(auto it = itl; it != itr; it++) it -> val += val;
    while(itl != itr) itl -> val += val, itl++;
}

int kth(int l, int r, int k){
    sit itr = split(r + 1), itl = split(l);
    vector<pair<int, int>> v;
    v.clear();
    for(sit it = itl; it != itr; it++) v.emplace_back(make_pair(it -> val, it -> r - it -> l + 1));
    sort(v.begin(), v.end());
    for(int i = 0; i < v.size(); i++){
        k -= v[i].second;
        if(k <= 0) return v[i].first;
    }
    return -1;
}

int binpow(int x, int y, int mod, int res = 1){
    for (x %= mod; y; y >>= 1, (x *= x) %= mod) if (y & 1) (res *= x) %= mod;
    return res;
}

int query(int l, int r, int x, int y){
    sit itr = split(r + 1), itl = split(l);
    int res(0);
    for(sit it = itl; it != itr; it++)
        res = (res + (it -> r - it -> l + 1) * binpow(it -> val, x, y)) % y;
    return res;
}

int n, m, vmax, seed;
int rnd() {
    int ret = (int)seed;
    seed = (seed * 7 + 13) % 1000000007;
    return ret;
}

inline void solve(){
    cin >> n >> m >> seed >> vmax;
    for(int i = 1; i <= n; i++){
```

```
            int a = rnd() % vmax + 1;
            s.insert(node{i, i, (int) a});
        }
        s.insert(node{n + 1, n + 1, 0});
        for(int i = 1; i <= m; i++){
            int l, r, x, y;
            int op = rnd() % 4 + 1;
            l = rnd() % n + 1, r = rnd() % n + 1;
            if(l > r) swap(l, r);
            if(op == 3) x = rnd() % (r - l + 1) + 1;
            else x = rnd() % vmax + 1;
            if(op == 4) y = rnd() % vmax + 1;
            if(op == 1) add(l, r, x);
            else if(op == 2) assign(l, r, x);
            else if(op == 3) cout << kth(l, r, x) << endl;
            else if(op == 4) cout << query(l, r, x, y) << endl;
        }
    }
```

15. 替罪羊树 (暴力重构二叉搜索树)

很暴力但是很牛逼的结构，平衡系数一般置为 0.75 即可。注意，二叉搜索树本质不变，所有二叉搜索树性质仍然适用。

平衡系数越大插入速度就越快，而访问和删除速度就会降低。反之则插入变慢

```
const int N = 2e5 + 10;
double alpha = 0.75;    //平衡系数

namespace SGT {
    struct node {
        int ls, rs;
        int w, wn, s, sz, sd;
    } tree[N];

    int cnt, root;

    //* recalculate the value of rt
    void calc(int rt) {
        tree[rt].s = tree[tree[rt].ls].s + tree[tree[rt].rs].s + 1;
        tree[rt].sz = tree[tree[rt].ls].sz + tree[tree[rt].rs].sz + tree[rt].wn;
        tree[rt].sd = tree[tree[rt].ls].sd + tree[tree[rt].rs].sd + (tree[rt].wn != 0);
    }

    //* determine if node k needs to be rebuild
    bool can_rebuild(int rt) {
        return tree[rt].wn && (alpha * tree[rt].s <= (double)max(tree[tree[rt].ls].s, tree[tree[rt].rs].s) ||
    ↪   (double)tree[rt].sd <= alpha * tree[rt].s);
    }

    int ldr[N];

    //* flatten the sub-tree of node rt in medium-order traversal
    void rebuild_nf(int &ldc, int rt) {
        if(!rt) return;
        rebuild_nf(ldc, tree[rt].ls);
        if(tree[rt].wn) ldr[ldc++] = rt;
        rebuild_nf(ldc, tree[rt].rs); // if the current node has been deleted, then no retained it.
    }

    //* rebuild the part [l, r] to a binary search tree(in array ldr)
    int rebuild_bd(int l, int r) {
        if(l >= r) return 0;
        int mid = l + r >> 1;   // choose the mid value as the root to make it balanced
        tree[ldr[mid]].ls = rebuild_bd(l, mid);
        tree[ldr[mid]].rs = rebuild_bd(mid + 1, r);
        calc(ldr[mid]);
        return ldr[mid];
    }

    //* rebuild the sub-tree of node rt
    void rebuild(int &rt) {
```

```
        int ldc = 0;
        rebuild_nf(ldc, rt);
        rt = rebuild_bd(0, ldc);
    }

    //* insert a node ot the subtree of rt with value k
    void insert(int &rt, int k) {
        if(!rt) {
            rt = ++cnt;
            if(!root) root = 1;
            tree[rt].w = k;
            tree[rt].ls = tree[rt].rs = 0;
            tree[rt].wn = tree[rt].s = tree[rt].sz = tree[rt].sd = 1;
        } else {
            if(tree[rt].w == k) tree[rt].wn++;
            else if(tree[rt].w < k) insert(tree[rt].rs, k);
            else insert(tree[rt].ls, k);
            calc(rt);
            if(can_rebuild(rt)) rebuild(rt);
        }
    }

    //* delete the node with value k from the sub-tree of k
    void del(int &rt, int k) {
        if(!rt) return;
        if(tree[rt].w == k) {
            if(tree[rt].wn) tree[rt].wn--;
        } else {
            if(tree[rt].w < k) del(tree[rt].rs, k);
            else del(tree[rt].ls, k);
        }
        calc(rt);
        if(can_rebuild(rt)) rebuild(rt);
    }

    //* find the lowest ranking with a weight strictly greater than k
    int upper_min(int rt, int k) {
        if(!rt) return 1;
        else if(tree[rt].w == k && tree[rt].wn)
            return tree[tree[rt].ls].sz + tree[rt].wn + 1;
        else if(tree[rt].w > k)
            return upper_min(tree[rt].ls, k);
        else
            return tree[tree[rt].ls].sz + tree[rt].wn + upper_min(tree[rt].rs, k);
    }

    //* find the maximum ranking with a weight strictly less than k
    int lower_max(int rt, int k) {
        if(!rt) return 0;
        else if(tree[rt].w == k && tree[rt].wn)
            return tree[tree[rt].ls].sz;
        else if(tree[rt].w < k)
            return tree[tree[rt].ls].sz + tree[rt].wn + lower_max(tree[rt].rs, k);
        else
            return lower_max(tree[rt].ls, k);
    }

    //* find the exactly value of rank k
    int getk(int rt, int k) {
        if(!rt) return 0;
        else if(tree[tree[rt].ls].sz < k && k <= tree[tree[rt].ls].sz + tree[rt].wn)
            return tree[rt].w;
        else if(tree[tree[rt].ls].sz + tree[rt].wn < k)
            return getk(tree[rt].rs, k - tree[tree[rt].ls].sz - tree[rt].wn);
        else
            return getk(tree[rt].ls, k);
    }

    inline int find_pre(int rt, int k) {
        return getk(rt, lower_max(rt, k));
    }
```

```cpp

    inline int find_aft(int rt, int k) {
        return getk(rt, upper_min(rt, k));
    }
}

using SGT::root;

inline void solve() {
    int n = 0; cin >> n;
    for(int i = 1; i <= n; i++) {
        int op, x; cin >> op >> x;
        if(op == 1) SGT::insert(root, x);
        else if(op == 2){
            SGT::del(root, x);
        }
        else if(op == 3) {
            int rnk = SGT::lower_max(root, x) + 1;
            cout << rnk << endl;
        } else if(op == 4) {
            int k = SGT::getk(root, x);
            cout << k << endl;
        } else if(op == 5) {
            int pre = SGT::find_pre(root, x);
            cout << pre << endl;
        } else if(op == 6) {
            int aft = SGT::find_aft(root, x);
            cout << aft << endl;
        }
    }
}
```

16.Splay

```cpp
#include <cstdio>
const int N = 100005;
int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];

struct Splay{
    void maintain(int x) { sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x]; }

    bool get(int x) { return x == ch[fa[x]][1]; }

    void clear(int x) {
        ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;
    }

    void rotate(int x){
        int y = fa[x], z = fa[y], chk = get(x);
        ch[y][chk] = ch[x][chk ^ 1];
        if (ch[x][chk ^ 1])  fa[ch[x][chk ^ 1]] = y;
        ch[x][chk ^ 1] = y;
        fa[y] = x, fa[x] = z;
        if (z) ch[z][y == ch[z][1]] = x;
        maintain(x), maintain(y);
    }

    void splay(int x){
        for (int f = fa[x]; f = fa[x], f; rotate(x))
            if (fa[f]) rotate(get(x) == get(f) ? f : x);
        rt = x;
    }

    void insert(int k){
        if (!rt){
            val[++tot] = k, cnt[tot]++;
            rt = tot;
            maintain(rt);
            return;
        }
        int cur = rt, f = 0;
```

```
        while (1){
            if (val[cur] == k){
                cnt[cur]++;
                maintain(cur), maintain(f);
                splay(cur);
                break;
            }
            f = cur, cur = ch[cur][val[cur] < k];
            if (!cur){
                val[++tot] = k, cnt[tot]++, fa[tot] = f;
                ch[f][val[f] < k] = tot;
                maintain(tot), maintain(f);
                splay(tot);
                break;
            }
        }
    }

    int rk(int k){
        int res = 0, cur = rt;
        while (1){
            if (k < val[cur]){
                cur = ch[cur][0];
            }
            else{
                res += sz[ch[cur][0]];
                if (k == val[cur]){
                    splay(cur);
                    return res + 1;
                }
                res += cnt[cur], cur = ch[cur][1];
            }
        }
    }

    int kth(int k){
        int cur = rt;
        while (1){
            if (ch[cur][0] && k <= sz[ch[cur][0]]) cur = ch[cur][0];
            else{
                k -= cnt[cur] + sz[ch[cur][0]];
                if (k <= 0){
                    splay(cur);
                    return val[cur];
                }
                cur = ch[cur][1];
            }
        }
    }

    int pre(){
        int cur = ch[rt][0];
        if (!cur) return cur;
        while (ch[cur][1]) cur = ch[cur][1];
        splay(cur);
        return cur;
    }

    int nxt(){
        int cur = ch[rt][1];
        if (!cur) return cur;
        while (ch[cur][0]) cur = ch[cur][0];
        splay(cur);
        return cur;
    }

    void del(int k){
        rk(k);
        if (cnt[rt] > 1) {
            cnt[rt]--;
            maintain(rt);
```

```
            return;
        }
        if (!ch[rt][0] && !ch[rt][1]){
            clear(rt);
            rt = 0;
            return;
        }
        if (!ch[rt][0]) {
            int cur = rt;
            rt = ch[rt][1], fa[rt] = 0;
            clear(cur);
            return;
        }
        if (!ch[rt][1]) {
            int cur = rt;
            rt = ch[rt][0], fa[rt] = 0;
            clear(cur);
            return;
        }
        int cur = rt, x = pre();
        fa[ch[cur][1]] = x;
        ch[x][1] = ch[cur][1];
        clear(cur);
        maintain(rt);
    }
} tree;

int main(){
    int n, opt, x;
    for (scanf("%d", &n); n; --n){
        scanf("%d%d", &opt, &x);
        if (opt == 1) tree.insert(x);
        else if (opt == 2) tree.del(x);
        else if (opt == 3) printf("%d\n", tree.rk(x));
        else if (opt == 4) printf("%d\n", tree.kth(x));
        else if (opt == 5) tree.insert(x), printf("%d\n", val[tree.pre()]), tree.del(x);
        else tree.insert(x), printf("%d\n", val[tree.nxt()]), tree.del(x);
    }
    return 0;
}
```

17.Link Cut Tree

(1). 无根树版本 (MakeRoot)

模板题：带修树上路径异或

```
#include <bits/stdc++.h>
#pragma gcc optimize("O2")
#pragma g++ optimize("O2")
#define int long long
#define endl '\n'
using namespace std;

const int N = 1e6 + 10;
int v[N];

namespace LCT{
    #define ls ch[x][0]
    #define rs ch[x][1]
    #define std_tag 0ll

    int ch[N][2], f[N], sum[N], val[N], tag[N], laz[N], siz[N];

    inline void push_up(int x) { sum[x] = sum[ls] ^ sum[rs] ^ v[x]; }

    inline void push(int x) { swap(ch[x][0], ch[x][1]), tag[x] ^= 1; }

    inline void push_down(int x) {
        if(tag[x]){
            if(ch[x][0]) push(ch[x][0]);
```

```
            if(ch[x][1]) push(ch[x][1]);
            tag[x] = 0;
        }
    }

    #define get(x) (ch[f[x]][1] == x)                           // 查询节点 X 是父亲的哪个儿子
    #define isRoot(x) (ch[f[x]][0] != x && ch[f[x]][1] != x) // 判断 X 是否是所在树的根

    inline void rotate(int x) { // 将 X 向上旋转一层
        int y = f[x], z = f[y], k = get(x);
        if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
        ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
        ch[x][!k] = y, f[y] = x, f[x] = z;
        push_up(y); push_up(x);
    }

    inline void update(int x) { // X 所在路径自上向下释放 Lazy 标记
        if(!isRoot(x)) update(f[x]);
        push_down(x);
    }

    inline void splay(int x) { // 将节点 X 旋至当前所在平衡树的树根 (带 LCT 性质认子不认父)
        update(x);
        for(int fa = f[x]; !isRoot(x); rotate(x), fa = f[x]){
            if(!isRoot(fa)) rotate(get(fa) == get(x) ? fa : x);
        }
        push_up(x);
    }

    int access(int x) { // 把从根到 X 的所有点放在一条实链里, 使根到 X 成为一条实路径
        int p;
        for(int p = 0; x; x = f[p = x]) splay(x), ch[x][1] = p, push_up(x);
        return p;
    }

    void makeRoot(int p) { access(p), splay(p), push(p); } // 使 X 点成为其所在树的根

    int findRoot(int x) { // 找到 X 所在树的根节点编号
        access(x), splay(x);
        while(ch[x][0]) push_down(x), x = ch[x][0];
        splay(x);
        return x;
    }

    void link(int x, int y) { // 在 x,y 之间连边
        makeRoot(x);
        if(findRoot(y) != x) f[x] = y;
    }

    void split(int x, int y) { // 提取出 x,y 之间的路径
        makeRoot(x);
        access(y), splay(y);
    }

    void cut(int x, int y){ // 删除 x,y 间的边
        makeRoot(x);
        if(findRoot(y) == x && f[y] == x && !ch[y][0]){
            f[y] = ch[x][1] = 0;
            push_up(x);
        }
    }
}

inline void solve(){
    int n, m; cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> v[i];
    while(m--){
        int op, x, y; cin >> op >> x >> y;
        switch (op) {
            case 0: LCT::split(x, y); cout << LCT::sum[y] << endl; break;
            case 1: LCT::link(x, y); break;
```

```
            case 2: LCT::cut(x, y); break;
            case 3: LCT::splay(x); v[x] = y;
        }
    }
}

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

(2). 有根树版本

模板题：QTREE7($multiset$ 维护子树最大值)

一棵树，每个点初始有个点权和颜色。

0 u：询问所有 u, v 路径上的最大点权, 要满足 u, v 路径上所有点的颜色都相同。

1 u：反转 u 的颜色。

2 u w：把 u 的点权改成 w 。

$color_i \in [0, 1],\ \ w_i \in [-10^9, 10^9],\ \ n, m \leq 10^5$ 。

```
#include <bits/stdc++.h>
#pragma gcc optimize("O2")
#pragma g++ optimize("O2")
#define int long long
#define elif else if
#define endl '\n'
using namespace std;

const int N = 2e5 + 10, MOD = 1e9 + 7;

int w[N], c[N], fa[N];

struct LCT{
    struct Info{
        int maxx;
        multiset<int> st;
    }tree[N];
    int ch[N][2], f[N], tag[N];

    inline void push_up(int x) {
        tree[x].maxx = max({tree[ch[x][0]].maxx, tree[ch[x][1]].maxx, w[x]});
        if(!tree[x].st.empty())
            tree[x].maxx = max(tree[x].maxx, *tree[x].st.rbegin());
    }

    inline void push(int x) { swap(ch[x][0], ch[x][1]), tag[x] ^= 1; }

    inline void push_down(int x) {
        if(tag[x]){
            if(ch[x][0]) push(ch[x][0]);
            if(ch[x][1]) push(ch[x][1]);
            tag[x] = 0;
        }
    }

    #define get(x) (ch[f[x]][1] == x)
    #define isRoot(x) (ch[f[x]][0] != x && ch[f[x]][1] != x)

    inline void rotate(int x){
        int y = f[x], z = f[y], k = get(x);
        if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
        ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
        ch[x][!k] = y, f[y] = x, f[x] = z;
        push_up(y), push_up(x);
```

```
    }

    inline void update(int x){
        while(!isRoot(x)) update(f[x]);
        push_down(x);
    }

    inline void splay(int x){
        // update(x);
        for(int fa = f[x]; !isRoot(x); rotate(x), fa = f[x]){
            if(!isRoot(fa)) rotate(get(fa) == get(x) ? fa : x);
        }
        push_up(x);
    }

    int access(int x){
        int p;
        for(p = 0; x; x = f[p = x]){
            splay(x);
            if(ch[x][1]) tree[x].st.insert(tree[ch[x][1]].maxx);
            ch[x][1] = p;
            if(ch[x][1]) tree[x].st.erase(tree[ch[x][1]].maxx);
            push_up(x);
        }
        return p;
    }

    int findRoot(int x) {
        access(x), splay(x);
        while(ch[x][0]) push_down(x), x = ch[x][0];
        splay(x);
        return x;
    }

    void link(int x, int y){
        if(!y) return;
        access(y), splay(y), splay(x);
        f[x] = y, ch[y][1] = x;
        push_up(y);
    }

    void cut(int x, int y){
        if(!y) return;
        access(x); splay(x);
        ch[x][0] = f[ch[x][0]] = 0;
        push_up(x);
    }
}lct[2];

vector<int> g[N];

void dfs(int u, int ufa){
    for(auto v : g[u]){
        if(v == ufa) continue;
        lct[c[v]].link(v, u), fa[v] = u;
        dfs(v, u);
    }
}

inline void solve(){
    int n = 0; cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    for(int i = 1; i <= n; i++) cin >> c[i];
    for(int i = 1; i <= n; i++) cin >> w[i];
    lct[0].tree[0].maxx = lct[1].tree[0].maxx = -2e9;
    dfs(1, 0);
    int q = 0; cin >> q;
```

```
    while(q--){
        int op = 0, u = 0; cin >> op >> u;
        if(op == 0) {
            int ufa = lct[c[u]].findRoot(u);
            if(c[ufa] == c[u]) cout << lct[c[u]].tree[ufa].maxx << endl;
            else cout << lct[c[u]].tree[lct[c[u]].ch[ufa][1]].maxx << endl;
        } elif (op == 1) {
            lct[c[u]].cut(u, fa[u]);
            c[u] ^= 1;
            lct[c[u]].link(u, fa[u]);
        } else {
            lct[c[u]].access(u);
            lct[c[u]].splay(u);
            cin >> w[u];
            lct[c[u]].push_up(u);
        }
    }

}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0);
    cout << fixed << setprecision(12);
    int t = 1; // cin >> t;
    while(t--) solve();
    return 0;
}
```

exSTL

优先队列

- binary_heap_tag
- pairing_heap_tag 支持修改
- thin_heap_tag 如果修改只有 increase 则较快，不支持 join

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

typedef __gnu_pbds::priority_queue<LL, less<LL>, pairing_heap_tag> PQ;
__gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::point_iterator it;
PQ pq, pq2;

int main() {
    auto it = pq.push(2);
    pq.push(3);
    assert(pq.top() == 3);
    pq.modify(it, 4);
    assert(pq.top() == 4);
    pq2.push(5);
    pq.join(pq2);
    assert(pq.top() == 5);
}
```

平衡树

- ov_tree_tag
- rb_tree_tag
- splay_tree_tag
- mapped: null_type 或 null_mapped_type（旧版本）为空
- Node_Update 为 tree_order_statistics_node_update 时才可以 find_by_order & order_of_key
- find_by_order 找 order + 1 小的元素（其实都是从 0 开始计数），或者有 order 个元素比它小的 key
- order_of_key 有多少个比 r_key 小的元素

- join & split

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using Tree = tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>;
Tree t;
```

持久化平衡树

```cpp
#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;

int main() {
    FOR (i, 0, 5) s.push_back(i); // 0 1 2 3 4
    s.replace(1, 2, s); // 0 (0 1 2 3 4) 3 4
    auto ss = s.substr(2, 2); // 1 2、
    s.erase(2, 2); // 0 1 4
    s.insert(2, s); // equal to s.replace(2, 0, s)
    assert(s[2] == s.at(2)); // 2
}
```

哈希表

```cpp
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

gp_hash_table<int, int> mp;
cc_hash_table<int, int> mp;
```

附录：ZKW 线段树 + 不带修 RMQ($O(N\log\log N)+O(1)$)

普通的 zkw 线段树解法分为线性建树和单次 $O(logn)$ 的查询，显然查询是瓶颈。

观察 zkw 线段树的查询结构：

```cpp
  for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
    if (~l & 1) ret = max(ret, zkw[l ^ 1]);
    if (r & 1) ret = max(ret, zkw[r ^ 1]);
  }
```

发现就是两边链查询，查询的信息类似于一条链上所有左/右儿子的兄弟们的最大值。

它是可以快速合并的，直接采用倍增优化（树上 ST 表），然后 $O(1)$ 回答询问。

由于树高 $O(logn)$，故预处理 ST 的时空复杂度均为 $O(nloglogn)$。

甚至可以原线段树都不要了，直接用数组建出 ST 表（见代码）。

综上，预处理时间复杂度：$O(n\log\log n)$，单次查询复杂度 $O(1)$，空间复杂度 $O(n\log\ \log\ n)$，优点是代码较短。

例题：第一行输入序列长度和查询数，接着一行输入序列，接着若干行每行一个查询区间，所有输入为 10^5 以内的正整数。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9, Mx = 1 << 17;
int n, m, M = 1, lg[Mx << 1];
pair<int, int> val[Mx << 1][5];
int main() {
  scanf("%d%d", &n, &m);
  while (M <= n) M <<= 1;
  for (int i = M + 1, a; i <= M + n; ++i) {
    scanf("%d", &a);
    ((i & 1) ? val[i ^ 1][0].first : val[i ^ 1][0].second) = a;
  }
  for (int i = M - 1; i > 1; --i)
    ((i & 1) ? val[i ^ 1][0].first : val[i ^ 1][0].second) =
        max(val[i << 1][0].first, val[i << 1 | 1][0].second);
  for (int i = 2; i < M + M; ++i) lg[i] = lg[i >> 1] + 1;
  for (int i = 2; i < M + M; ++i)
```

```
      for (int h = 1; h < 5; ++h) {
        val[i][h].first =
            max(val[i][h - 1].first, val[i >> (1 << (h - 1))][h - 1].first);
        val[i][h].second =
            max(val[i][h - 1].second, val[i >> (1 << (h - 1))][h - 1].second);
      }
    for (int l, r, len; m; --m) {
      scanf("%d%d", &l, &r);
      l += M - 1, r += M + 1, len = lg[l ^ r];
      printf("%d\n", max(max(val[l][lg[len]].first,
                             val[l >> (len - (1 << lg[len]))][lg[len]].first),
                         max(val[r][lg[len]].second,
                             val[r >> (len - (1 << lg[len]))][lg[len]].second)));
    }
    return 0;
}
```