数据结构

笛卡尔树

小根笛卡尔树

```
cin >> n;
for (int i = 0;i < n;++i)cin >> nums[i];
for (int i = 0;i < n;++i)rs[i] = -1;
for (int i = 0;i < n;++i)ls[i] = -1;
top = 0;
for (int i = 0; i < n; i++) {
    int k = top;
    while (k > 0 && nums[stk[k - 1]] > nums[i]) k--;
    if (k) rs[stk[k - 1]] = i;  // rs代表笛卡尔树每个节点的右儿子
    if (k < top) ls[i] = stk[k];  // ls代表笛卡尔树每个节点的左儿子
    stk[k++] = i;
    top = k;
}
```

线段树

```
struct Info {
    Info operator+ (const Info& x) {

    }
}tree[MAXN << 2];

int n;
void update(int p, int x, int l = 1, int r = n, int i = 1) {
    if (l > r) return;
    if (p < l || r < p) return;
    if (l == r) {
        tree[i] = Info();
        return;
    }
    int mid = l + r >> 1;
    update(p, x, l, mid, i << 1);
    update(p, x, mid + 1, r, i << 1 | 1);
    tree[i] = tree[i << 1] + tree[i << 1 | 1];
}

Info query(int ql, int qr, int l = 1, int r = n, int i = 1) {
    if (l > r) return Info();
    if (qr < l || r < ql)return Info();
    if (ql <= l && r <= qr)return tree[i];
    int mid = l + r >> 1;
    return query(ql, qr, l, mid, i << 1) + query(ql, qr, mid + 1, r, i << 1 | 1);
}
```

```cpp
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
```

```
        modify(2 * p + 1, m, r, x, v);
    }
    pull(p);
}
void modify(int p, const Info &v) {
    modify(1, 0, n, p, v);
}
Info rangeQuery(int p, int l, int r, int x, int y) {
    if (l >= y || r <= x) {
        return Info();
    }
    if (l >= x && r <= y) {
        return info[p];
    }
    int m = (l + r) / 2;
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
}
Info rangeQuery(int l, int r) {
    return rangeQuery(1, 0, n, l, r);
}
template<class F>
int findFirst(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int res = findFirst(2 * p, l, m, x, y, pred);
    if (res == -1) {
        res = findFirst(2 * p + 1, m, r, x, y, pred);
    }
    return res;
}
template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 0, n, l, r, pred);
}
template<class F>
int findLast(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) {
        return -1;
```

```
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
struct Info {
    int cnt = 0;
    i64 sum = 0;
    i64 ans = 0;
};
Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.ans = a.ans + b.ans + a.cnt * b.sum - a.sum * b.cnt;
    return c;
}
```

lazy tag

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
```

```
    apply(2 * p, tag[p]);
    apply(2 * p + 1, tag[p]);
    tag[p] = Tag();
}
void modify(int p, int l, int r, int x, const Info &v) {
    if (r - l == 1) {
        info[p] = v;
        return;
    }
    int m = (l + r) / 2;
    push(p);
    if (x < m) {
        modify(2 * p, l, m, x, v);
    } else {
        modify(2 * p + 1, m, r, x, v);
    }
    pull(p);
}
void modify(int p, const Info &v) {
    modify(1, 0, n, p, v);
}
Info rangeQuery(int p, int l, int r, int x, int y) {
    if (l >= y || r <= x) {
        return Info();
    }
    if (l >= x && r <= y) {
        return info[p];
    }
    int m = (l + r) / 2;
    push(p);
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
}
Info rangeQuery(int l, int r) {
    return rangeQuery(1, 0, n, l, r);
}
void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
    if (l >= y || r <= x) {
        return;
    }
    if (l >= x && r <= y) {
        apply(p, v);
        return;
    }
```

```
    int m = (l + r) / 2;
    push(p);
    rangeApply(2 * p, l, m, x, y, v);
    rangeApply(2 * p + 1, m, r, x, y, v);
    pull(p);
}
void rangeApply(int l, int r, const Tag &v) {
    return rangeApply(1, 0, n, l, r, v);
}
template<class F>
int findFirst(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    push(p);
    int res = findFirst(2 * p, l, m, x, y, pred);
    if (res == -1) {
        res = findFirst(2 * p + 1, m, r, x, y, pred);
    }
    return res;
}
template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 0, n, l, r, pred);
}
template<class F>
int findLast(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    push(p);
    int res = findLast(2 * p + 1, m, r, x, y, pred);
    if (res == -1) {
        res = findLast(2 * p, l, m, x, y, pred);
    }
```

```
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    i64 a = 0, b = 0;
    void apply(Tag t) {
        a = std::min(a, b + t.a);
        b += t.b;
    }
};

int k;

struct Info {
    i64 x = 0;
    void apply(Tag t) {
        x += t.a;
        if (x < 0) {
            x = (x % k + k) % k;
        }
        x += t.b - t.a;
    }
};
Info operator+(Info a, Info b) {
    return {a.x + b.x};
}
```

Fenwick Tree 树状数组

```cpp
template<typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n + 5, T{});
    }

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

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

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int select(const T& k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i] <= k) {
                x += i;
                cur = cur + a[x];
            }
```

```
        }
        return x;
    }
};
```

dsu并查集

```
struct dsu {
    std::vector<int> d;
    dsu(int n) { d.resize(n); iota(d.begin(), d.end(), 0); }
    int get_root(int x) { return d[x] = (x == d[x] ? x : get_root(d[x])); };
    bool merge(int u, int v) {
        if (get_root(u) != get_root(v)) {
            d[get_root(u)] = get_root(v);
            return true;
        }
        else return false;
    }
};
```

对顶堆

```
namespace Set {
    const int kInf = 1e9 + 2077;
    std::multiset<int> less, greater;
    void init() {
        less.clear(), greater.clear();
        less.insert(-kInf), greater.insert(kInf);
    }
    void adjust() {
        while (less.size() > greater.size() + 1) {
            std::multiset<int>::iterator it = (--less.end());
            greater.insert(*it);
            less.erase(it);
        }
        while (greater.size() > less.size()) {
            std::multiset<int>::iterator it = greater.begin();
            less.insert(*it);
            greater.erase(it);
        }
    }
    void add(int val_) {
        if (val_ <= *greater.begin()) less.insert(val_);
        else greater.insert(val_);
        adjust();
    }
    void del(int val_) {
        std::multiset<int>::iterator it = less.lower_bound(val_);
        if (it != less.end()) {
            less.erase(it);
        }
        else {
            it = greater.lower_bound(val_);
            greater.erase(it);
        }
        adjust();
    }
    int get_middle() {
        return *less.rbegin();
    }
}
```