0x50 数据结构

0x51 并查集

```cpp
struct DSU
{
    vector<int> p, sz;
    DSU(int n) : p(n + 1), sz(n + 1, 1) { iota(p.begin(), p.end(), 0); }
    int find(int x)
    {
        if (p[x] != x) p[x] = find(p[x]);
        return p[x];
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y)
    {
        int px = find(x), py = find(y);
        if (px == py) return false;

        sz[px] += sz[py];
        p[py] = px;
        return true;
    }
    int size(int x) { return sz[find(x)]; }
};
```

0x52 树状数组

```cpp
template <typename T>
struct Fenwick
{
    const int n;
    vector<T> tr;
    Fenwick(int n) : n(n), tr(n + 1) {}

    int lowbit(int x) { return x & -x; }
    void add(int x, T c)
    {
        for (int i = x; i <= n; i += lowbit(i)) tr[i] += c;
    }
    T sum(int x)
    {
        T res = 0;
        for (int i = x; i; i -= lowbit(i)) res += tr[i];
        return res;
    }
    T range_sum(int l, int r) { return sum(r) - sum(l - 1); }
};
```

0x53 线段树

权值线段树

```
// 求区间最大值
template<class Info, typename T>
struct SegmentTree
{
    const int n;
    vector<Info> tr;
    SegmentTree(int n) : n(n), tr(4 * n) {}
    SegmentTree(const vector<T>& init) : SegmentTree(init.size() - 1)
    {
        function<void(int, int, int)> build = [&](int u, int l, int r)
        {
            if (l == r)
            {
                tr[u].x = init[r];
                return;
            }

            int mid = l + r >> 1;
            build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
            pushup(u);
        };
        build(1, 1, n);
    }

    void pushup(Info& u, Info& l, Info& r)
    {
        u.x = max(l.x, r.x);
    }
    void pushup(int u) { pushup(tr[u], tr[u << 1], tr[u << 1 | 1]); }

    void modify(int u, int l, int r, int x, const T &v)
    {
        if (l == r)
        {
            tr[u].x = v;
            return;
        }

        int mid = l + r >> 1;
        if (x <= mid) modify(u << 1, l, mid, x, v);
        else modify(u << 1 | 1, mid + 1, r, x, v);
        pushup(u);
    }
    void modify(int x, const T &v) { modify(1, 1, n, x, v); }

    Info rangeQuery(int u, int l, int r, int x, int y)
    {
        if (l >= x && r <= y) return tr[u];

        int mid = l + r >> 1;
        if (y <= mid) return rangeQuery(u << 1, l, mid, x, y);
        else if (x > mid) return rangeQuery(u << 1 | 1, mid + 1, r, x, y);
        else
        {
            auto left = rangeQuery(u << 1, l, mid, x, y);
            auto right = rangeQuery(u << 1 | 1, mid + 1, r, x, y);
            Info res;
            pushup(res, left, right);
            return res;
        }
    }
    Info rangeQuery(int l, int r) { return rangeQuery(1, 1, n, l, r); }
};

struct Info  // 注意long long 和 MLE
{
    int x;
    Node(int v = 0) : x(v) {}
};

SegmentTree<Info, int> seg(n);  // 传大小
SegmentTree<Info, int> seg(a);  // 传数组
```

懒标记线段树

```
// 求区间和
template<class Info, typename T>
struct LazySegmentTree
{
    const int n;
    vector<Info> tr;
    LazySegmentTree(int n) : n(n), tr(4 * n) {}
    LazySegmentTree(const vector<T>& init) : LazySegmentTree(init.size() - 1)
    {
        function<void(int, int, int)> build = [&](int u, int l, int r)
        {
            if (l == r)
            {
                tr[u].x = init[r];
                return;
            }

            int mid = l + r >> 1;
            build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
            pushup(u);
        };
        build(1, 1, n);
    }

    void pushup(Info& u, Info& l, Info& r)
    {
        u.x = l.x + r.x;
        u.len = l.len + r.len;
    }
    void pushup(int u) { pushup(tr[u], tr[u << 1], tr[u << 1 | 1]); }

    void apply(Info& t, const T &tag)
    {
        t.x += tag * t.len;
        t.tag += tag;
    }

    void pushdown(Info& u, Info& l, Info& r)
    {
        apply(l, u.tag);
        apply(r, u.tag);
        u.tag = 0;
    }
    void pushdown(int u) { pushdown(tr[u], tr[u << 1], tr[u << 1 | 1]); }

    void modify(int u, int l, int r, int x, const T &tag)
    {
        if (l == r)
        {
            apply(tr[u], tag);
            return;
        }

        pushdown(u);
        int mid = l + r >> 1;
        if (x <= mid) modify(u << 1, l, mid, x, tag);
        else modify(u << 1 | 1, mid + 1, r, x, tag);
        pushup(u);
    }
    void modify(int x, const T &tag) { modify(1, 1, n, x, tag); }

    void rangeApply(int u, int l, int r, int x, int y, const T &tag)
    {
        if (l >= x && r <= y)
        {
            apply(tr[u], tag);
            return;
        }

        pushdown(u);
        int mid = l + r >> 1;
        if (x <= mid) rangeApply(u << 1, l, mid, x, y, tag);
        if (y > mid) rangeApply(u << 1 | 1, mid + 1, r, x, y, tag);
        pushup(u);
    }
    void rangeApply(int l, int r, const T &tag) { rangeApply(1, 1, n, l, r, tag);
```

```
}

    Info rangeQuery(int u, int l, int r, int x, int y)
    {
        if (l >= x && r <= y) return tr[u];

        pushdown(u);
        int mid = l + r >> 1;
        if (y <= mid) return rangeQuery(u << 1, l, mid, x, y);
        else if (x > mid) return rangeQuery(u << 1 | 1, mid + 1, r, x, y);
        else
        {
            auto left = rangeQuery(u << 1, l, mid, x, y);
            auto right = rangeQuery(u << 1 | 1, mid + 1, r, x, y);
            Info res;
            pushup(res, left, right);
            return res;
        }
    }
    Info rangeQuery(int l, int r) { return rangeQuery(1, 1, n, l, r); }
};

struct Info  // 注意long long 和 MLE
{
    int x;
    int tag, len;   // 懒标记，区间长度
    Info(int v = 0, int c = 1) : x(v), tag(v), len(c) {}
};
```

0x54 可持久化线段树 / 主席树

```
struct PersistentSegmentTree
{
    struct Info
    {
        int l, r, cnt;
    };
    vector<Info> tr;
    vector<int> root;
    int idx = 0;
    PersistentSegmentTree(int n) : tr(n + 1 << 5), root(n + 1)
    {
        function<void(int &, int, int)> build = [&](int &u, int l, int r)
        {
            u = ++ idx;
            if (l == r) return ;
            int mid = l + r >> 1;
            build(tr[u].l, l, mid), build(tr[u].r, mid + 1, r);
        };
        build(root[0], 1, n);
    }
    void insert(int pre, int &now, int l, int r, int x)
    {
        now = ++ idx;
        tr[now] = tr[pre];
        tr[now].cnt ++;

        if (l == r) return ;

        int mid = l + r >> 1;
        if (x <= mid) insert(tr[pre].l, tr[now].l, l, mid, x);
        else insert(tr[pre].r, tr[now].r, mid + 1, r, x);
    }
    int query(int pre, int now, int l, int r, int k)
    {
        if (l == r) return l;

        int mid = l + r >> 1;
        int sum = tr[tr[now].l].cnt - tr[tr[pre].l].cnt;
        if (k <= sum) return query(tr[pre].l, tr[now].l, l, mid, k);
        else return query(tr[pre].r, tr[now].r, mid + 1, r, k - sum);
    }
};
```

0x57 最近公共祖先

性质

1. $LCA(\{u\}) = u$。
2. u 是 v 的祖先，当且仅当 $LCA(u,v) = u$。
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $LCA(u,v)$ 的两棵不同子树中。
4. 前序遍历中，$LCA(S)$ 出现在所有 S 中元素之前，后序遍历中 $LCA(S)$ 则出现在所有 S 中元素之后。
5. 两点集并的最近公共祖先为两点集分别的最近公共祖先的最近公共祖先，即 $LCA(A \cup B) = LCA(LCA(A), LCA(B))$ 。
6. 两点的最近公共祖先必定处在树上两点间的最短路上。
7. $dis(u,v) = dep(u) + dep(v) - 2dis(LCA(u,v))$ ，其中 dis 是树上两点间的距离，dep 代表某点到树根的距离（即深度）。

倍增

```
vector<int> g[N];
int dep[N], fa[N][20];

void bfs(int root)
{
    queue<int> q;
    memset(dep, 0x3f, sizeof dep);

    dep[0] = 0, dep[root] = 1;
    q.push(root);

    while (!q.empty())
    {
        auto x = q.front(); q.pop();

        for (auto y : g[x])
        {
            if (dep[y] <= dep[x] + 1) continue;

            dep[y] = dep[x] + 1;
            fa[y][0] = x;
            q.push(y);

            for (int i = 1; i < 20; i ++ ) fa[y][i] = fa[fa[y][i - 1]][i - 1];
        }
    }
}

int lca(int x, int y)
{
    if (dep[x] < dep[y]) swap(x, y);

    for (int i = 19; i >= 0; i -- )
        if (dep[fa[x][i]] >= dep[y])
            x = fa[x][i];

    if (x == y) return x;

    for (int i = 19; i >= 0; i -- )
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];

    return fa[x][0];    // 返回节点
}
```