4.7 线段树分裂

```
/*
0 p x y: 分裂，将可重集合p中[x, y]的元素移动到一个新的可重集合中（可重集合编号从1开始，每次+1）
1 p t: 合并，将可重集合t的信息合并进可重集合p
2 p x q：单点加，可重集合p，加入x个q
3 p x y: 区间查询，可重集合p，查询[x, y]
4 p k：查询第k小，可重集合p，查询第k小
*/
struct ST {
    struct Node {
        int s;
    };
    vector<Node> t;
    vector<int> lc, rc;
    int N;

    ST(int n) {
        t.resize(n + 1);
        lc.resize(n + 1);
        rc.resize(n + 1);
        N = 0;
    }

    #define LC (lc[x])
    #define RC (rc[x])

    void pushup(int x) {
        t[x].s = t[LC].s + t[RC].s;
    }
    void build(int &x, int l, int r, const vector<int>& a) {
        x = ++N;
        if(l == r) {
            t[x].s = a[l];
            return;
        }

        int mid = l + r >> 1;
        build(LC, l, mid, a);
        build(RC, mid+1, r, a);
        pushup(x);
    }
    void add(int &x, int l, int r, int pos, int va) {
        if(!x) x = ++N;
        if(pos <= l && r <= pos) {
            t[x].s += va;
            return;
        }
        if(pos < l || r < pos) return;
```

```
        int mid = l + r >> 1;
        if(pos <= mid) add(LC, l, mid, pos, va);
        else add(RC, mid+1, r, pos, va);
        pushup(x);
    }
    int query(int x, int l, int r, int L, int R) {
        if(x == 0) return 0;
        if(L <= l && r <= R) return t[x].s;
        if(R < l || r < L) return 0;

        int mid = l + r >> 1;
        return query(LC, l, mid, L, R) + query(RC, mid+1, r, L, R);
    }
    int kth(int x, int l, int r, int k) {
        if(l == r) return l;
        if(t[x].s < k) return -1;
        int mid = l + r >> 1;
        if(t[LC].s >= k) return kth(LC, l, mid, k);
        else return kth(RC, mid+1, r, k - t[LC].s);
    }
    void split(int &x, int &y, int l, int r, int L, int R) {
        if(x == 0) return;
        if(L <= l && r <= R) {
            y = x;
            x = 0;
            return;
        }
        if(R < l || r < L) return;

        y = ++N;

        int mid = l + r >> 1;
        split(LC, lc[y], l, mid, L, R);
        split(RC, rc[y], mid+1, r, L, R);
        pushup(x);
        pushup(y);
    }
    int merge(int x, int y, int l, int r) {
        if(x == 0 || y == 0) return x + y;
        if(l == r) { t[x].s += t[y].s; return x; }

        int mid = l + r >> 1;
        LC = merge(LC, lc[y], l, mid);
        RC = merge(RC, rc[y], mid+1, r);
        pushup(x);
        return x;
    }
};

void work() {
    int n, m;
    cin >> n >> m;
```

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

    int N = 0;
    vector<int> rt(m + 1);

    ST st(n * 20);
    st.build(rt[++N], 1, n, a);

    for(int i = 1; i <= m; i++) {
        int opt, p, x;
        cin >> opt >> p >> x;
        if(opt == 0) {
            int y;
            cin >> y;
            st.split(rt[p], rt[++N], 1, n, x, y);
        } else if(opt == 1) {
            rt[p] = st.merge(rt[p], rt[x], 1, n);
        } else if(opt == 2) {
            int y;
            cin >> y;
            st.add(rt[p], 1, n, y, x);
        } else if(opt == 3) {
            int y;
            cin >> y;
            cout << st.query(rt[p], 1, n, x, y) << '\n';
        } else if(opt == 4) {
            cout << st.kth(rt[p], 1, n, x) << '\n';
        } else assert(false);
    }
}
```