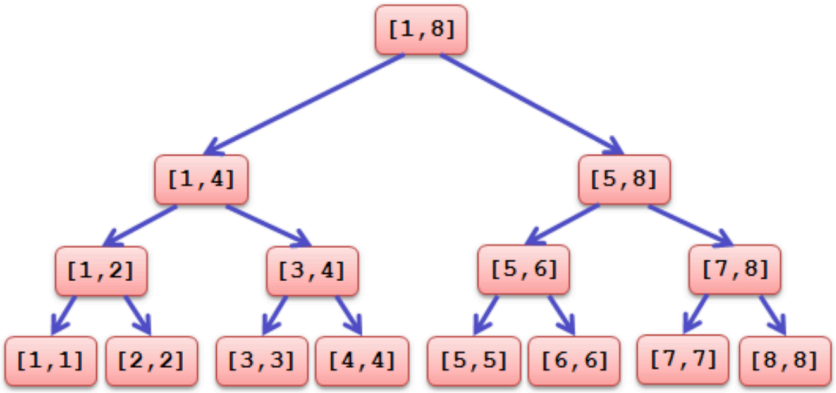
线段树

Segment Tree

Idea：递归地二分区间使之成为树的结构。



Complexity：单次操作 $O(\lg n)$

Code（双标记）：

```
#define lid id<<1
#define rid id<<1|1
#define len(id) (tr[id].r - tr[id].l + 1)
#define mid ((tr[id].l + tr[id].r) >> 1)
struct segTree{
    int l, r;
    LL sum, add, mul;
}tr[N<<2];
inline void pushup(int id){
    tr[id].sum = tr[lid].sum + tr[rid].sum;
}
inline void pushdown(int id){
    if(tr[id].mul != 1 && tr[id].l != tr[id].r){
        (tr[lid].mul *= tr[id].mul) %= p;
        (tr[lid].add *= tr[id].mul) %= p;
        (tr[lid].sum *= tr[id].mul) %= p;
        (tr[rid].mul *= tr[id].mul) %= p;
        (tr[rid].add *= tr[id].mul) %= p;
        (tr[rid].sum *= tr[id].mul) %= p;
        tr[id].mul = 1;
    }
    if(tr[id].add != 0 && tr[id].l != tr[id].r){
        (tr[lid].add += tr[id].add) %= p;
        (tr[lid].sum += len(lid) * tr[id].add % p) %= p;
        (tr[rid].add += tr[id].add) %= p;
        (tr[rid].sum += len(rid) * tr[id].add % p) %= p;
        tr[id].add = 0;
    }
}
void build(int id, int l, int r){
    tr[id].l = l; tr[id].r = r;
    tr[id].sum = tr[id].add = 0;
    tr[id].mul = 1;
    if(tr[id].l == tr[id].r){
        tr[id].sum = c[l] % p;
```

```
        return;
    }
    build(lid, l, mid);
    build(rid, mid+1, r);
    pushup(id);
}
void add(int id, int l, int r, LL v){
    pushdown(id);
    if(tr[id].l == l && tr[id].r == r){
        (tr[id].add += v) %= p;
        (tr[id].sum += len(id) * v % p) %= p;
        return;
    }
    if(r <= mid)
        add(lid, l, r, v);
    else if(l > mid)
        add(rid, l, r, v);
    else{
        add(lid, l, mid, v);
        add(rid, mid+1, r, v);
    }
    pushup(id);
}
void mul(int id, int l, int r, LL v){
    pushdown(id);
    if(tr[id].l == l && tr[id].r == r){
        (tr[id].mul *= v) %= p;
        (tr[id].add *= v) %= p;
        (tr[id].sum *= v) %= p;
        return;
    }
    if(r <= mid)
        mul(lid, l, r, v);
    else if(l > mid)
        mul(rid, l, r, v);
    else{
        mul(lid, l, mid, v);
        mul(rid, mid+1, r, v);
    }
    pushup(id);
}
LL querySum(int id, int l, int r){
    pushdown(id);
    if(tr[id].l == l && tr[id].r == r)
        return tr[id].sum % p;
    if(r <= mid)
        return querySum(lid, l, r) % p;
    else if(l > mid)
        return querySum(rid, l, r) % p;
    else
        return (querySum(lid, l, mid) + querySum(rid, mid+1, r)) % p;
}
```