线段树 v1.2

万能的网络参考资料

B站视频1

B站视频2

参考博客1

参考博客2

典型例题

现在有一个长度$n = 10^5$的数组，我们需要进行以下的操作m次

1. 给下标在L~R范围内的数字都加上一个A
2. 查询下标在L~R范围内的总和
   $1 \leq n, m \leq 10^5, 1 \leq L \leq R \leq n$

分析

- 如果没有更新操作，直接前缀和即可
- 如果直接在数组上暴力的操作，复杂度爆炸：
  - 更新复杂度$O(n)$
  - 查询复杂度$O(n)$
  - 总时间复杂度$O(nm)$
- 所以我们需要使用线段树的结构，来优化查询和更新的操作

线段树支持的几种基本操作

1. 建树，复杂度$O(nlogn)$
2. 更新，复杂度$O(logn)$
   - 包括：
   - 单点赋值
   - 单点增减
   - 区间赋值
   - 区间增减
   - 所有的单点操作都可以使用区间操作
3. 区间查询，复杂度$O(logn)$
   - 最大值

    - 最小值
    - 区间之和
    - GCD
    - 等等....

用线段树统计的东西,必须符合区间加法，否则，不可能通过分成的子区间来得到$[L, R]$的统计结果。

符合区间的例子：

- 数字之和 —— 总数字之和=左区间数字之和+有区间数字之和
- GCD —— 总GCD=gcd(左区间gcd，右区间gcd)
- 最大值 —— 总最大值=max(左区间最大值，右区间最大值)
- ...

不符合区间加法的例子：

- 众数 —— 只知道左右区间的众数,没法求总区间的众数
- 01序列的最长连续0 —— 只知道左右区间的最长连续0，没有办法知道总的最长连续0。**(但是可以通过其他的属性来辅助实现，记录每个节点lmx， rmx， mx 分别表示左端开始的最长连续0， 右端开始的最长连续0， 整个区间最长连续0)**
- ...

模板

> glj 的线段树模板魔改版，已经把线段树封装好了，可以当做类调用。
> 模板包括区间加法，区间求和，区间最大最小值，但做题的时候要根据具体的题目，修改内容实现。
> 参考代码
>
> A - HDU1166 - 敌兵布阵
>
> B - HDU1754 - I Hate It

```
struct segt {
    ll *a;
    struct Tree {
        int l, r;
        ll sum, max, min;
        ll lazy; // 延迟标记
        // 节点整体更新
        void update(ll v) {
            sum += v * (r - l + 1);
            lazy += v;
```

```
            max += v;
            min += v;
        }
        // 修改整一个节点[l, r]的值都为v
        void setnum(ll v) {
            sum = v;
            max = v;
            min = v;
        }
    } tree[maxn * 4]; // 一般开4倍大小的空间
    // 等价于 p * 2
    inline int lc(int p) {return p << 1;}
    // 等价于 p * 2 + 1
    inline int rc(int p) {return p << 1 | 1;}
    void modify(ll *arr) {
        a = arr;
    }
    // 向上更新
    void pushup(int p) {
        tree[p].sum = tree[lc(p)].sum + tree[rc(p)].sum;
        tree[p].max = max(tree[lc(p)].max, tree[rc(p)].max);
        tree[p].min = min(tree[lc(p)].min, tree[rc(p)].min);
    }
    // 向下传递标记
    void pushdown(int p) {
        if (tree[p].lazy == 0) return;
        tree[lc(p)].update(tree[p].lazy);
        tree[rc(p)].update(tree[p].lazy);
        tree[p].lazy = 0;
    }
    // 建树
    void build(int p, int l, int r) {
        tree[p].l = l, tree[p].r = r;
        tree[p].sum = tree[p].max = tree[p].min = tree[p].lazy = 0;
        if (l == r) {
            tree[p].sum = tree[p].max = tree[p].min = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(lc(p), l, mid);
        build(rc(p), mid + 1, r);
        pushup(p);
```

```cpp
}
// 单点更新 点x
void updateOne(int p, int x, ll v) {
    int L = tree[p].l, R = tree[p].r;
    if (L == x && L == R) {
        tree[p].setnum(v);
        return;
    }
    int mid = (L + R) >> 1;
    if (x <= mid) updateOne(lc(p), x, v);
    else  updateOne(rc(p), x, v);
    pushup(p);
}
// 实现区间[l, r] 加 v
void updateAdd(int p, int l, int r, ll v) {
    int L = tree[p].l, R = tree[p].r;
    if (l <= L && r >= R) {
        tree[p].update(v);
        return;
    }
    // 向下传递标记
    pushdown(p);
    int mid = (L + R) >> 1;
    if (l <= mid) {
        updateAdd(lc(p), l, r, v);
    }
    if (r > mid) {
        updateAdd(rc(p), l, r, v);
    }
    pushup(p);
}
// 查询区间[l, r]的和
ll querySum(int p, int l, int r) {
    int L = tree[p].l, R = tree[p].r;
    int mid = (L + R) >> 1;
    if (l <= L && r >= R) {
        return tree[p].sum;
    }
    // 向下传递标记
    pushdown(p);
    ll res = 0;
    if (l <= mid) {
```

```
            res += querySum(lc(p), l, r);
        }
        if (r > mid) {
            res += querySum(rc(p), l, r);
        }
        return res;
    }
    // 查询区间[l, r]的最大值
    ll queryMax(int p, int l, int r) {
        int L = tree[p].l, R = tree[p].r;
        int mid = (L + R) >> 1;
        if (l <= L && r >= R) {
            return tree[p].max;
        }
        // 向下传递标记
        pushdown(p);
        ll res = -INF;
        if (l <= mid) {
            res = max(res, queryMax(lc(p), l, r));
        }
        if (r > mid) {
            res = max(res, queryMax(rc(p), l, r));
        }
        return res;
    }
    // 查询区间[l, r]的最小值
    ll queryMin(int p, int l, int r) {
        int L = tree[p].l, R = tree[p].r;
        int mid = (L + R) >> 1;
        if (l <= L && r >= R) {
            return tree[p].min;
        }
        // 向下传递标记
        pushdown(p);
        ll res = INF;
        if (l <= mid) {
            res = min(res, queryMin(lc(p), l, r));
        }
        if (r > mid) {
            res = min(res, queryMin(rc(p), l, r));
        }
        return res;
```

```cpp
    }

    // 如果需要求多个值，可以使用全局变量，定义以下的内容
    // 并且一定要注意在使用前初始化
    ll SUM, MAX, MIN
    void query(int p, int l, int r) {
        int L = tree[p].l, R = tree[p].r;
        int mid = (L + R) >> 1;
        if (l <= L && r >= R) {    // 要查询的区间 包括了改区间
            SUM += tree[p].sum;
            MAX = max(MAX, tree[p].max);
            MIN = min(MIN, tree[p].min);
            return;
        }
        pushdown(p);
        ll res = INF;
        if (l <= mid) query(lc(p), l, r);
        if (r > mid) query(rc(p), l, r);
    }
}seg;
```