线段树

参考资料

线段树视频 1
线段树视频 2
参考博客 1
参考博客 2

例题

现在有一个长 $n = 10^5$ 的数组,我们有以下几种操作

1. 给下标在 L ~ R 范围的数字加上 A
2. 查询下标在 L ~ R 范围的数字的和

$1 \leq L \leq R \leq n$
操作共有 $m = 10^5$ 次

分析

- 如果没有更新操作,直接前缀和即可
- 如果直接在数组上操作,则复杂度爆炸
    - 更新复杂度 $O(n)$
    - 查询复杂度 $O(n)$
    - 总复杂度 $O(nm)$
- 所以我们使用线段树来优化更新和查询操作

线段树主要有以下几种基本操作:

1. 建树 $O(N\log_2 N)$
2. 更新 $O(\log_2 N)$
    - 单点赋值
    - 单点加
    - 区间赋值

- 区间加
- (所有单点操作均可用区间操作实现)

3. 区间查询 $O(\log_2 N)$
    - 最大值
    - 最小值
    - 和

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。

符合区间加法的例子：

- 数字之和——总数字之和 = 左区间数字之和 + 右区间数字之和
- 最大公因数(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序列的最长连续零——只知道左右区间的最长连续零，没法知道总的最长连续零(可以通过添加其他属性来实现,记录每个节点lmx,rmx,mx,表示从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
> 模板包含区间加,区间求和,区间求最大最小值,但做题的时候要根据具体情况再修改部分内容
>
> 参考代码
> A - 敌兵布阵 HDU - 1166
> B - I Hate It HDU - 1754

```
struct segt {
  ll *a;
  struct Tree {
    int l, r;
    ll  sum, lz, max, min;
    void update(ll v) {
      sum += v * (r - l + 1);
```

```
    lz  += v;
    max += v;
    min += v;
  }
} tree[N * 4];

void modify(ll *arr) {
  a = arr;
}

void pushup(int x) {
  tree[x].sum = tree[2 * x].sum + tree[2 * x + 1].sum;
  tree[x].max = max(tree[2 * x].max, tree[2 * x + 1].max);
  tree[x].min = min(tree[2 * x].min, tree[2 * x + 1].min);
}

void pushdown(int x) {
  if (tree[x].lz != 0) {
    tree[2 * x].update(tree[x].lz);
    tree[2 * x + 1].update(tree[x].lz);
    tree[x].lz = 0;
  }
}

// 建树
void build(int x, int l, int r) {
  tree[x].l   = l;
  tree[x].r   = r;
  tree[x].sum = tree[x].max = tree[x].min = tree[x].lz = 0;

  if (l == r) {
    tree[x].sum = tree[x].max = tree[x].min = a[l];

    return;
  }
  int mid = (l + r) / 2;
  build(2 * x,     l,       mid);
  build(2 * x + 1, mid + 1, r);
  pushup(x);
}

// 区间l-r加c
void updateADD(int x, int l, int r, ll c) {
  int L = tree[x].l, R = tree[x].r;
  int mid = (L + R) / 2;
```

```
    if ((l <= L) && (r >= R)) {
      tree[x].update(c);
      return;
    }

    pushdown(x);

    if (l <= mid) updateADD(2 * x, l, r, c);

    if (r > mid) updateADD(2 * x + 1, l, r, c);
    pushup(x);
  }

  // 查询区间 l-r 的和
  ll querySUM(int x, int l, int r) {
    int L = tree[x].l, R = tree[x].r;
    int mid = (L + R) / 2;
    ll  res = 0;

    if ((l <= L) && (r >= R)) { // 要更新区间包括了该区间
      return tree[x].sum;
    }

    pushdown(x);

    if (l <= mid) res += querySUM(2 * x, l, r);

    if (r > mid) res += querySUM(2 * x + 1, l, r);
    pushup(x);
    return res;
  }

  // 查询区间 l-r 的最大值
  ll queryMAX(int x, int l, int r) {
    int L = tree[x].l, R = tree[x].r;
    int mid = (L + R) / 2;
    ll  res = -INF;

    if ((l <= L) && (r >= R)) { // 要更新区间包括了该区间
      return tree[x].max;
    }

    pushdown(x);

    if (l <= mid) res = max(res, queryMAX(2 * x, l, r));
```

```
    if (r > mid) res = max(res, queryMAX(2 * x + 1, l, r));
    pushup(x);
    return res;
  }

  // 查询区间 l-r 的最小值
  ll queryMIN(int x, int l, int r) {
    int L = tree[x].l, R = tree[x].r;
    int mid = (L + R) / 2;
    ll  res = INF;

    if ((l <= L) && (r >= R)) { // 要更新区间包括了该区间
      return tree[x].min;
    }

    pushdown(x);

    if (l <= mid) res = min(res, queryMIN(2 * x, l, r));

    if (r > mid) res = min(res, queryMIN(2 * x + 1, l, r));
    pushup(x);
    return res;
  }

  // 如果要求多个值，可以用全局变量
  // 使用前记得将 SUM, MAX, MIN 初始化。
  ll SUM, MAX, MIN;
  void query(int x, int l, int r) {
    int L = tree[x].l, R = tree[x].r;
    int mid = (L + R) / 2;

    if ((l <= L) && (r >= R)) { // 要更新区间包括了该区间
      SUM += tree[x].sum;
      MAX = max(MAX, tree[x].max);
      MIN = min(MIN, tree[x].min);
      return;
    }

    pushdown(x);

    if (l <= mid) query(2 * x, l, r);

    if (r > mid) query(2 * x + 1, l, r);
    pushup(x);
  }
};
```