算法讲堂

主讲人：刘文越

树状数组

树状数组是一个查询和修改复杂度都为$O(logn)$的数据结构。主要用于数组的**单点修改**和**区间查询**。

关于树状数组的存储结构其实只有一个处理数组。（a 为原数组，tr 为处理后数组，n 为数据范围）

```
long long a[N], tr[N], n;
```

接下来，介绍 $lowbit$ 操作，该操作返回当前数组二进制下最后的 “1” 所在的位置。

```
inline void lowbit(int x) {
    return x & -x;
}
```

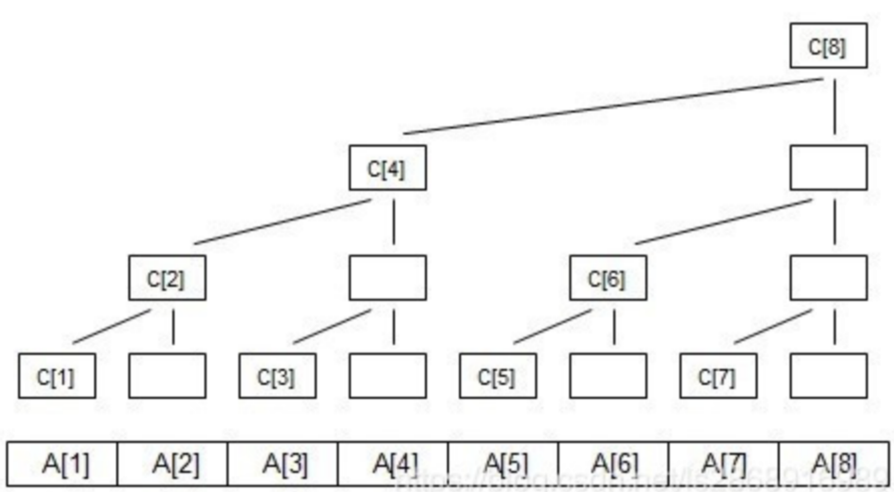
修改操作

```
inline void add(int pos, long long x) {
    for (long long i = pos; i <= n; i += lowbit(i)) tr[i] += x;
}
```

查询操作

```
long long getSum(long long x) {
    long long rs = 0;
    for (long long i = x; i; i -= lowbit(i)) rs += tr[i];
    return rs;
}
```

感谢CSDN上萧何山大佬博客 [树状数组详解] 中的图：



$tr[i]$ 代表子树的叶子节点的权值之和，如图可以知道：

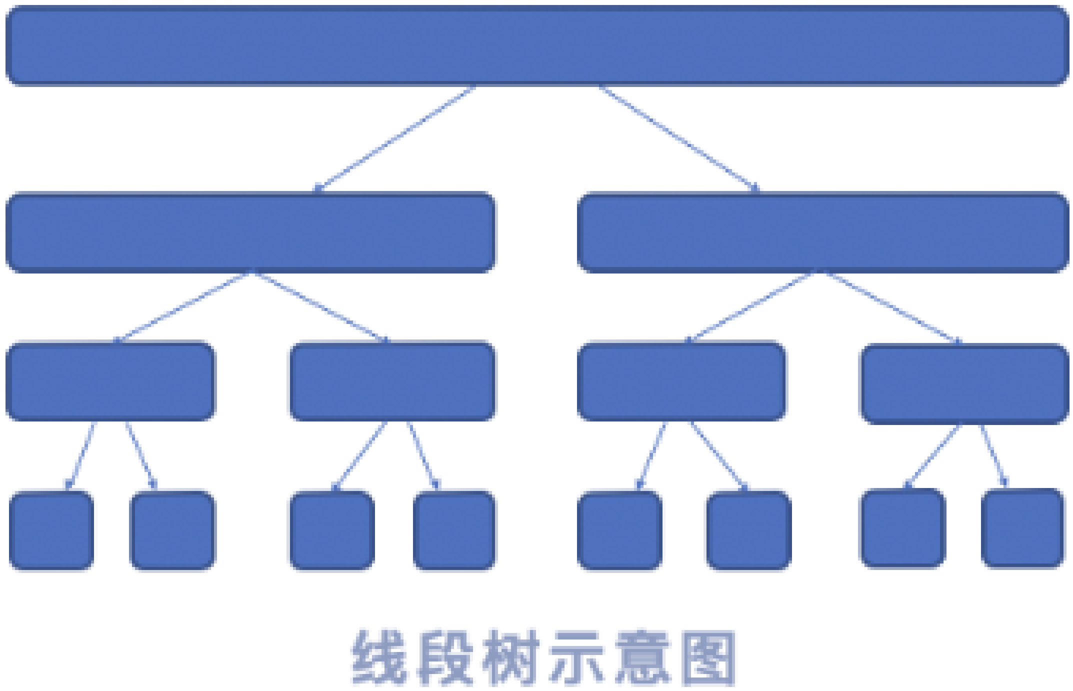
$$
\begin{aligned}
&tr[1] = A[1];\\
&tr[2] = A[1] + A[2];\\
&tr[3] = A[3];\\
&tr[4] = A[1] + A[2] + A[3] + A[4];\\
&tr[5] = A[5];\\
&tr[6] = A[5] + A[6];\\
&tr[7] = A[7];\\
&tr[8] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6] + A[7] + A[8];
\end{aligned}
$$

单点修改就是对每个包含该节点的树状数组节点进行更新，通过 $+= lowbit(i)$ 向上维护。
区间查询则是通过将 $[1, (\text{所求下标})]$ 这个区间依据**二进制**划分后相加求解。

线段树

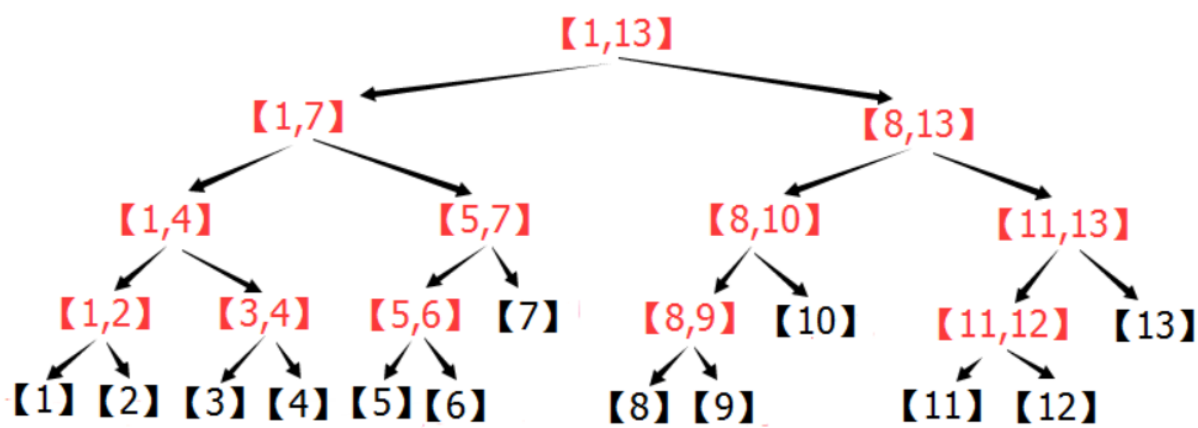
树状数组实现了 $O(logn)$ 级别的区间查询和单点修改操作，但是如果我们需要**区间修改/查询操作**的时候，应该怎么办呢？

当然是线段树，伟大的线段树之神会赐福于一切的数据维护。



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。

而每个相同的 n 区间分解的结果都是相同的。



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——$总GCD = gcd(左区间GCD,右区间GCD)$
- 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

线段树定义：

```
long long w[N], n, m;
struct Node {
    int l, r;
    long long sum, add;
} tr[N << 2];
```

pushup操作：

```
void pushup(int u) {
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}
```

build操作：

```
void build(int u, int l, int r) {
    if (l == r) tr[u] = {l, r, w[l], 0};
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
```

pushdown操作：

```
void pushdown(int u) {
    Node &rt = tr[u], &lp = tr[u << 1], &rp = tr[u << 1 | 1];
    if (rt.add) {
        lp.add += rt.add, lp.sum += (LL)(lp.r - lp.l + 1) * rt.add;
        rp.add += rt.add, rp.sum += (LL)(rp.r - rp.l + 1) * rt.add;
        rt.add = 0;
    }
}
```

修改和查询：

```
void modify(int u, int l, int r, int v) {
    if (l <= tr[u].l && tr[u].r <= r) {
        tr[u].sum += (tr[u].r - tr[u].l + 1) * v;
        tr[u].add += v;
    } else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, v);
        if (r > mid) modify(u << 1 | 1, l, r, v);
        pushup(u);
```

```
    }
}
long long query(int u, int l, int r) {
    if (l <= tr[u].l && tr[u].r <= r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    long long rs = 0;
    if (l <= mid) rs = query(u << 1, l, r);
    if (r > mid) rs += query(u << 1 | 1, l, r);
    return rs;
}
```

题目：

（1）线段树

（2）树状数组

推荐博客：
线段树详解 （原理，实现与应用）