树状数组

树状数组的提出

有四种数组处理问题：单点修改，单点查询；单点修改，区间查询；区间修改，单点查询；区间修改，区间查询。

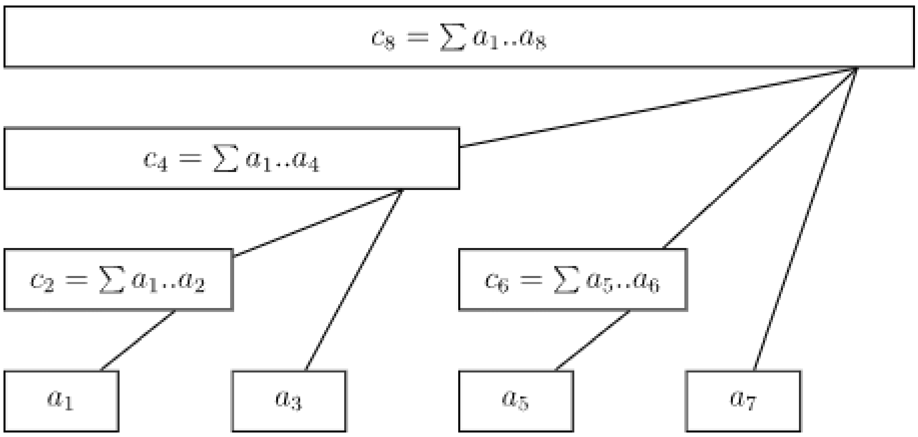
对普通的数组进行区间求和我们可以用前缀和数组，进行区间修改（指在数组某一段区间上加或减一个值）我们可以用差分数组。但是像上述四种将查询与修改结合起来的情况，我们不得不舍弃这两种方式，因为前缀和数组的更新操作太过复杂，比如更改数组第一个元素，整个前缀和数组都要改；差分数组的查询操作太过复杂，比如查询最后一个值，我们需要将整个差分数组进行相加。

于是树状数组横空出世，给我们带来修改与查询结合情况下的新思路。

要注意的是，原汁原味的树状数组能够做到的是单点修改和区间查询，如果要做到区间修改的话，只需要加一点数学上的处理即可，代码是一样的。

树状数组的性质（可跳过直接看代码）

下图中，a是原数组，c是树状数组



列举出来：

$$c_1 = a_1$$

$$c_2 = a_1 + a_2$$

$$c_3 = a_3$$

$$c_4 = a_1 + a_2 + a_3 + a_4$$

$$c_5 = a_5$$

$$c_6 = a_5 + a_6$$

$$c_7 = a_7$$

$$c_8 = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8$$

观察树状数组的性质，有如下几点：

- 树状数组与原数组等长
- 每个树状数组的元素是若干原数组元素的加和，且这若干的原数组元素是连续的（即每个树状数组元素都是原数组的某一区间和）
- c_i 的最后一个加数一定是a_i
- 当 i 为奇数的时候，$c_i = a_i$，当 i 是2的次幂的时候，$c_i = a_1 + \ldots + a_i$
- （这点不容易看出来）对于任何 i ,都有$c_i = a_{i-2^k+1} + a_{i-2^k+2} + \ldots + a_i$,其中，k 是 i的二进制表示中从最低位往高位的连续0的数量

  比如，$6 = 0110$,从低位向高位数有1个0，所以$k = 1i$ 。又比如,$8 = 1000$,从低位向高位数有3个0，所以$k = 3$

  这里直接给出求k 的方法：$k = i\&(-i)$。我们将这个k 称为$lowbit(i)$, 意为 i 低位0的个数。
- a_i 包含于若干个树状数组的元素中，分别是c_i, $c_{i+2^{k1}}$, $c_{(i+2^{k1})+2^{k2}} \cdots$一直到数组最大长度。这里的$k_1$是 i 对应的 k, k_2 是$i + 2^{k_1}$对应的 k ,以此类推
- 原数组的前 i项和为$c_i + c_{i-2^{k1}} + c_{(i-2^{k1})-2^{k2}+\ldots}$，一直到下标为1 。这里的$k_1$, k_2解释同上一条性质

树状数组的代码

树状数组在不理解原理的情况下也能使用，因为它的代码实在是太简单了

树状数组有两种基本操作：单点更新，求前 n 项和

```
int lowbit(int x)//在下面两种操作里用到的一个临时计算函数
{
    return x & (-x);
}

void update(int x, int k)//实现a[i]+=k
{
    while (x <= n) {
        c[x] += k;
        x += lowbit(x);
    }
}

int getsum(int x)//求前x项和
{
    int ans = 0;
    while (x > 0) {
        ans += c[x];
        x -= lowbit(x);
    }
    return ans;
}
```

区间更新，单点查询

上述方法只能实现原数组的单点更新，如果是区间更新的话我们可以采用差分建树，即不用原数组建树，使用差分数组建树。这样我们可以用两次单点更新实现区间更新，使用前 i 项和实现单点查询，十分简单

区间更新，区间查询

我们假设 d 是 a 的差分数组。

则 a 的前 i 项和为：

$a_1 + a_2 + a_3 + ... + a_i = (d_1) + (d_1 + d_2) + (d_1 + d_2 + d_3) + ... + (d_1 + d_2 + d_3 + .. + d_i) = n(d_1 + d_2 + ... + d_i) - (0d_1 + 1d_2 + 2d_3 + ... + (i-1)d_i)$

所以我们建两个树状数组，一个是差分建树，另一个用$(i-1)d_i$建树即可。