FZOI

信息学

树状数组

【问题描述】

给定n个数a[1],a[2],a[3],...,a[n]，现在有下面两种操作：

1.询问区间 [x , y] 的和，并输出。

2.将下标为x的数增加val。

一共进行m次操作。

1<=n，m<=100000，保证每个数在 int 范围内。

方法一：暴力枚举

定义数组a存储n个数。求区间和的时间复杂度为O(n)，将a[x]增加val的时间复杂度为O(1)，总时 间复杂度为O(nm)。

方法二：前缀和

定义数组sum，表示前缀和。求区间和的时间复杂度为O(1)，将a[x]增加val的时间复杂度为O(n)，因为每进行增加操作，就需要更新所有前缀和，总时间复杂度为O(nm)。

为什么两种方法的时间复杂度都这么高呢?

第一种方法，数组a的元素存储的信息只包含一个数，管的太少，所以求和慢。

第二种方法，数组sum的元素存储的信息包含了前面的所有数，管的太多，导致修改数值时牵扯到 的元素很多，所以修改慢。

因此，那么我们就找一个数组存储的信息包含的数不多，也不少就可以了，这就是——树状数组。

不太多，也不太少这种思想，其实刚好是树状数组的神奇之处。这也是程序设计中的一种思路，取折中后最后的，因此会有这种复杂度O(logN)，O($\sqrt{N}$)，都是在几个操作的极限情况下，找最佳平衡方案。

基本思想：

任意正整数可以写成关于2的不重复次幂相加的形式。

若正整数x=21,二进制表示为10101，$x=2^4+2^2+2^0$。

对于区间[1,x]，可以分解成log(x)个小区间：

(1)长度为2^4的小区间：$[1,2^4]$。 区间结尾：10000

(2)长度为2^2的小区间：$[2^4+1, 2^4+2^2]$。 区间结尾：10100

(3)长度为2^0的小区间：$[2^4+2^2+1, 2^4+2^2+2^0]$。 区间结尾：10101

分解出的小区间有个共同特点：

若区间结尾为y，则**区间长度=y的二进制下最小的2次幂**。

定义**lowbit(x):**

表示x(x为非负整数)的二进制下的最小2次幂。即x的二进制最低位的1以及后面的0构成的值。

求lowbit(x)：

设x的第k位为1，第0~k-1位都是0。

先把x取反，此时第k为变为0,第0~k-1位都为1。

再令x=x+1，此时因为进位，第k位变为1，第0~k-1位都为0。同时，因为取反操作，第k+1位到最高位都与原来相反。

再进行与运算。n&(~n+1)

原码、反码和补码

- 原码:指一个二进制数左边加上符号位后所得到的码，且当二进制数大于0时，符号位为0；二进制数小于0时，符号位为1；二进制数等于0时，符号位可以为0或1。
- 反码：正数的反码与其原码相同；负数的反码是对其原码逐位取反，但符号位除外。
- 补码：正数的补码与原码相同，负数的补码是其对应正数二进制所有位取反后加1。

在计算机中通常使用补码进行储存。

在补码表示下，~x=-1-x，因此lowbit（x)=x&(~x+1)=x&(-x)

求解lowbit(x)

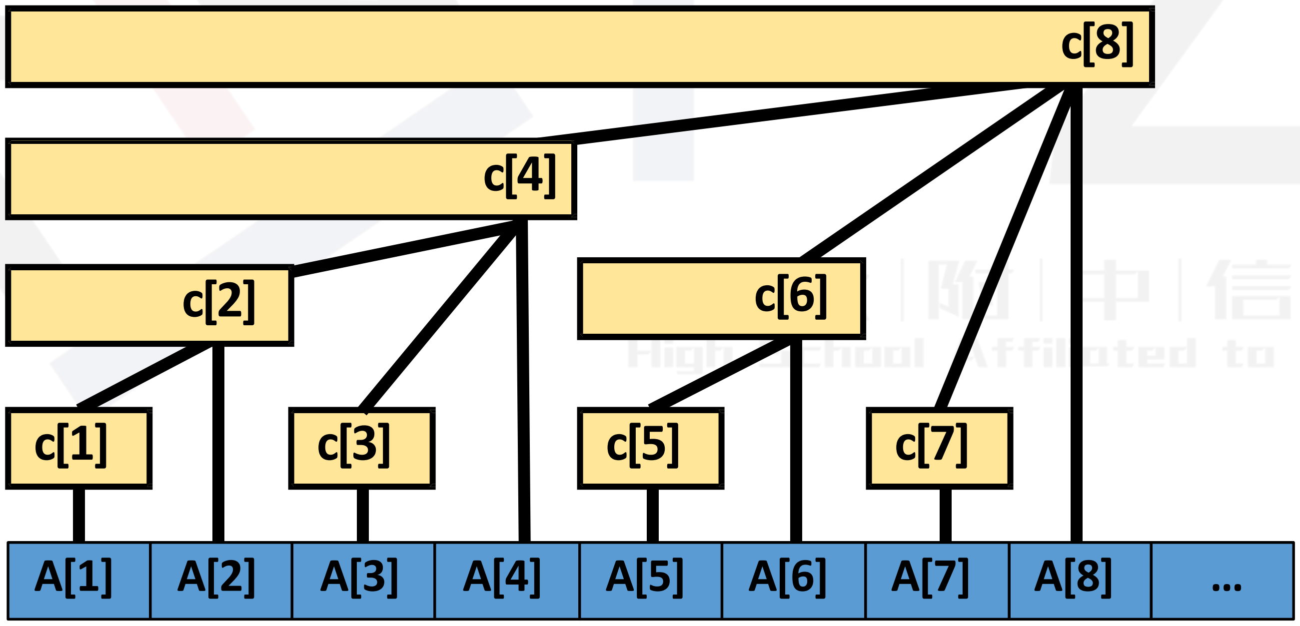

```
int lowbit(x)
{
        return x&(-x);
}
```

```
int lowbit(x)
{
        return x&(~x+1);
}
```

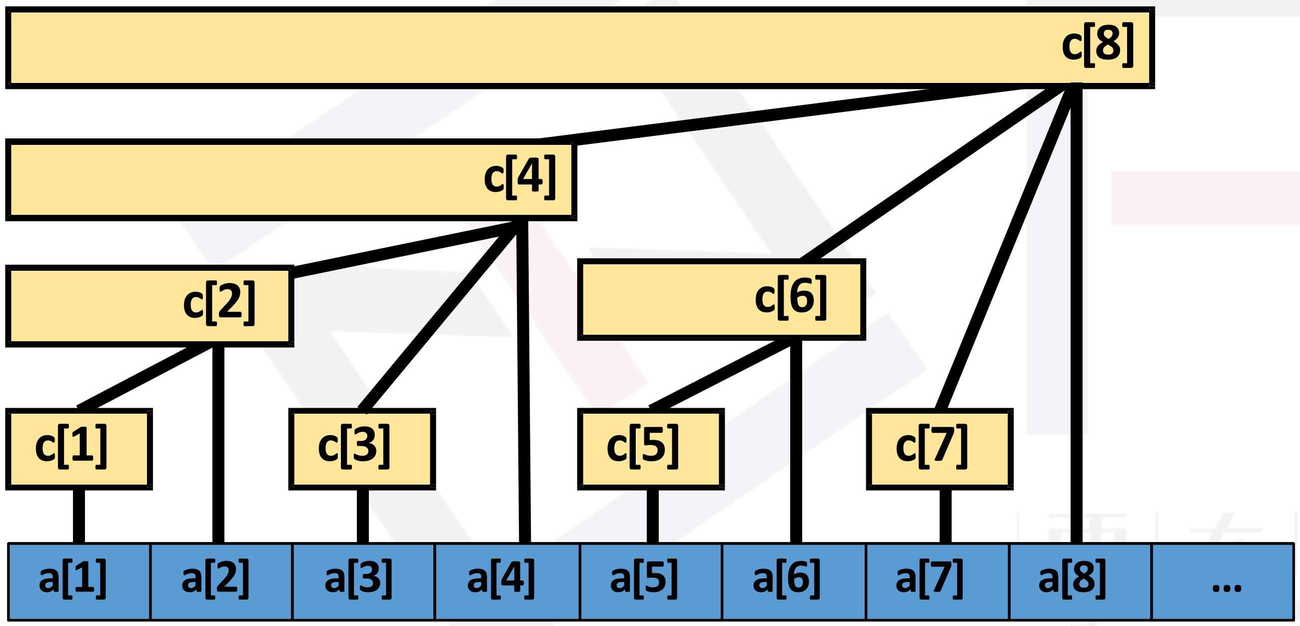
基本算法：

给定序列A，我们建立一个数组c，其中c[x]保存序列A区间长度为lowbit(x)的区间和，即区间[x-lowbit(x)+1,x]的所有数的和。

数组c可以看作一个树形结构，图中最下面一行是N个叶子结点，N=8，表示A[1~8]。 如果N不是2的整次幂，那么树状数组就是一个具有同样结构的森林。



树状数组



x	x的二进制	区间长度 lowbit(x)	表示区间的和
1	1	1	[a[1] ,a[1]]
2	10	2	[a[1] ,a[2]]
3	11	1	[a[3] ,a[3]]
4	100	4	[a[1] ,a[4]]
5	101	1	[a[5] ,a[5]]
6	110	2	[a[5] ,a[6]]
7	111	1	[a[7] ,a[7]]
8	1000	8	[a[1] ,a[8]]

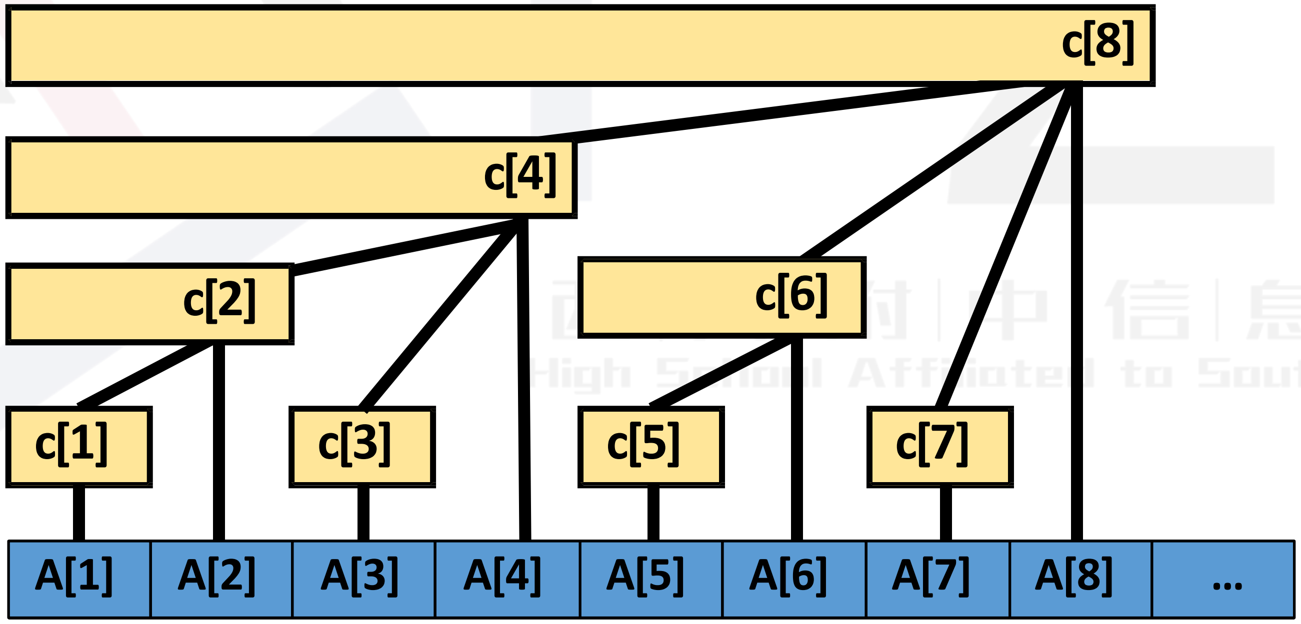
该结构满足：

① 每个内部结点c[x]保存以它为根的子树中所有叶结点的和。

② 每个内部结点c[x]的子结点个数等于lowbit(x)的大小

③ 除树根外，每个内部结点**c[x]的父亲为c[x+lowbit(x)]**

④ 树的深度为O(log N)



1.对某个元素进行加法(单点增加)

给序列中的A[x]增加val。同时需要维护c[]表示的区间和。

包含A[x]的只有结点c[x]以及其所有祖先结点，因此逐一对这些结点进行操作。任意结点最多有logN个祖先结点。

因此单点增加的时间复杂度为**O(logN)**。

```
void  update(int x,int y)     //a[x]增加val
{
            for(int i=x;i<=n;i+=lowbit(i))          //i的父结点为i+lowbit(i)
                        c[i]+=val;
}
```

2.区间查询

询问区间[1,x]的和。

按照最初**区间拆分**的方法，[1,x]被拆分成了最多logN个小区间，每个小区间的区间和都保存在了c[]。

例如：求解[1,21]，21的二进制表示为10101，包含区间c[10101],c[10100],c[10000]

```
int  sum(int x)      //求前缀和a[1]~a[x]
{
        int ans=0;
        for(int i=x;i>0;i-=lowbit(i))
                ans+=c[i];
        return ans;
}
```

求区间[x,y]：

sum(y)-sum(x-1)

3.建树状数组

c[]初始化为0，每输入一个数，就update(x,A[x])。

时间复杂度为O(NlogN)

```
for(int i=1;i<=n;i++)
{
        cin>>a;
        update(i,a);
}
```

树状数组能够处理的下标为1~n，不能出现下标为0的情况，lowbit(0)=0会陷入死循环。

因此， 如果出现下标为0的情况，可以全部右移一个单位。

树状数组扩展——初始化

建树状数组 ，c[ ] 初 始 化 为 0，每输入一个数，就update(x,A[x])。

时间复杂度为O(NlogN)

优化：

c[x]保存序列A区间长度为lowbit(x)的 区 间 和， 即 区 间 A[**x-lowbit(x)+1]~A[x]**的 所有数的和。

时间复杂度为O(N)

前缀和求解

树状数组与逆序对

利用树状数组也可以求逆序对：

数组c[x]：表示**数值**[x-lowbit(x)+1,x]出现的数的个数。

1. 在序列A的**数值范围**内建立树状数组，初始值全为0。

2. **倒序**扫描给定的序列A，对于每一个A[i]：

在树状数组中查询前缀和1~A[i]-1的值，即出现在A[i]后面，且比A[i]小的数的个数，组成逆序对，加入到答案ans中。

执行单点增加，数值A[i]增加1，表示A[i]又出现了一次。

3.ans就是答案。

数值范围较大时，需要进行离散化。

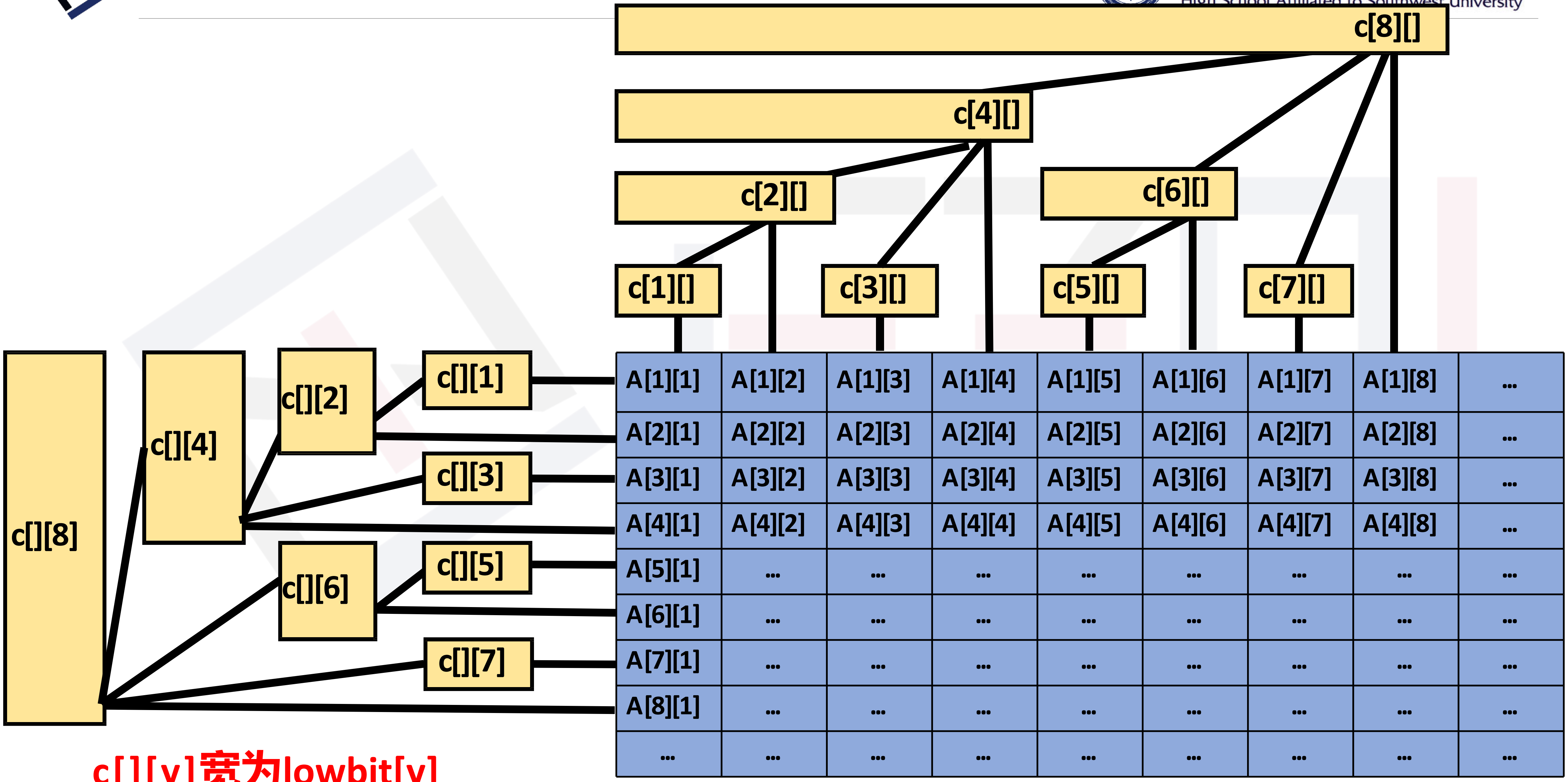
树状数组也能够在二维数组上也可以应用。

在一维树状数组中， c[x]代表的是记录区间尾为x ，长度为lowbit(x)的区间和。

所以在二维树状数组当中，定义c[x][y]记录的是右下角为(x,y)，长 为lowbit(x),宽为lowbit(y)的区间和。

所有单点修改和区间查询的操作就改成了二维的了。

c[x][]长为lowbit[x]



c[][y]宽为lowbit[y]

二维树状数组

某坐标增加，询问某区间数量，变成二维了。

```
void  update(int x,int y,int val)          //a(x,y)增加val
{
        for(int i=x;i<=n;i+=lowbit(i))
                for(int j=y;j<=m;j+=lowbit(j))
                        c[i][j]+=val;
}
```

```
int sum(int x,int y) //左上角(1,1)，右下角(x,y)的矩阵和
{
        int ans=0;
        for(int i=x;i<=n;i-=lowbit(i))
                for(int j=y;j<=m;j-=lowbit(j))
                        ans+=c[i][j];

        return ans;
}
```

树状数组扩展——求前缀最大值/最小值

使用c[x] 维护区间结尾为 a[x]， 长度为 lowbit(x)的最大值。

```
void update(int x,int val)         //将a[x]更新为val，更新c数组最大值
{
    for(int i=x;i<=n;i+=lowbit(i))          //i的父结点为i+lowbit(i)
        c[i]=max(c[i],val)
}
```

```
int Max(int x)                     //a[1]~a[x]的最大值
{
    int ans=0;
    for(int i=x;i>0;i-=lowbit(i))
        ans=max(ans,c[i]);
    return ans;
}
```

树状数组扩展——区间修改

对区间a[x]~a[y]的所有数整体增加val。

逐个进行单点修改?

时间复杂度为：NlogN

区间修改能够减少操作次数吗?

差分数组

区间修改

如果存在序列a的差分数组d，对区间[x,y]增加val，可以视为差分数组： d[x]+=val，d[y+1]-=val

单点查询

求修改后的a[x]的值。

实际上就是差分数组的前缀和。

区间查询

求数组$a[1{:}n]$的前缀和　　再维护 $d[i]\times i$ 的前缀和

$$a[1]+a[2]+\cdots+a[i]=d[1]\times i+d[2]\times(i-1)+\cdots+d[i]\times 1$$
$$=[d[1]\times(i+1)+d[2]\times(i+1)+\cdots+d[i]\times(i+1)]-[d[1]\times 1+d[2]\times 2+\cdots+d[i]\times i]$$

$$=(i+1)\times(d[1]+d[2]+\cdots+d[i])-(d[1]\times 1+d[2]\times 2+\cdots+d[i]\times i)$$

树状数组通过二进制来维护区间来实现。

树状数组维护**前缀信息**：前缀和，前缀最大值等。