一个小问题

这里有n个数，$a_1,a_2…a_n$，也可以当作数组a有n个元素

实现两种操作，执行q次：

- 修改某个数的值
- 求出某段区间的和

n<=100000,q<=100000

方法1

开一个100000的数组,题目让我干啥我干啥.
时间复杂度:单次修改O(1),单次求和最坏O(n),总时间复杂度最坏为$O(n^2)$,会超时.

为什么超时呢?

因为每次求和的速度太慢了。

方法2

开一个1..100000的数组c，，c[i]存储的是$a_1+a_2+…+a_i$，
其实就是前缀和
要求$a_i..a_j$的和，那么就是$c_j-c_{(i-1)}$。
显然，这下，单次求和的复杂度就变成了O(1)。
但是单次修改的最坏复杂度变成了O(n)。
总复杂度又变成了$O(n^2)$
看看我们这两次失败的尝试，原因在哪里

第一次，我们数组元素a_i存储的信息只包含那个a_i，管的太少了，所以求和起来慢。

第二次，我们数组元素c_i存储的信息包含了$a_1,a_2…a_i$,管的太多了...这样会导致修改a值的时候涉及到的元素太多了

容易想到,我们如果能想到一种方法,让c数组元存储的a的数目适当多,就可以了

信息学

树状数组

西南大学附属中学校

信息奥赛教练组

树状数组

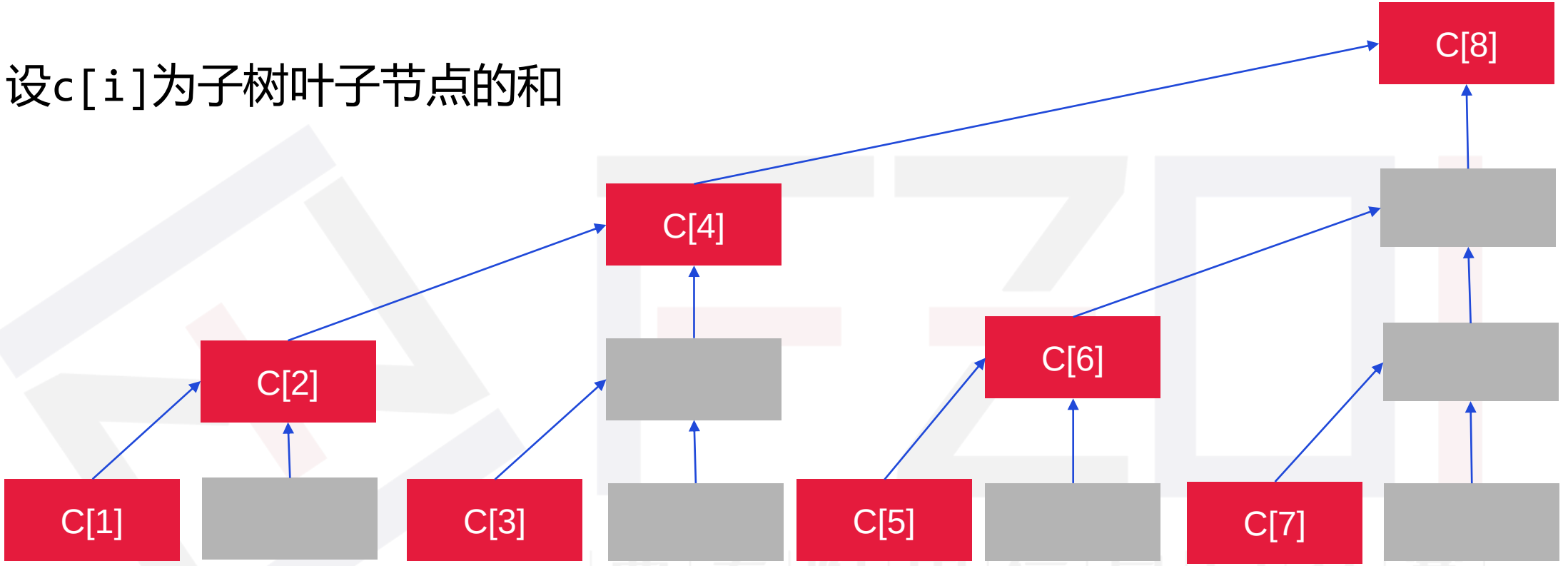
树状数组是一种新的存储方式，顾名思义树一样形状的存储.

每个c_i存储的a元素数目不是一开始规定好的,而是根据i的不同而不同的

进而达到什么目的呢?
下面看下形象的解释

树状数组

设c[i]为子树叶子节点的和



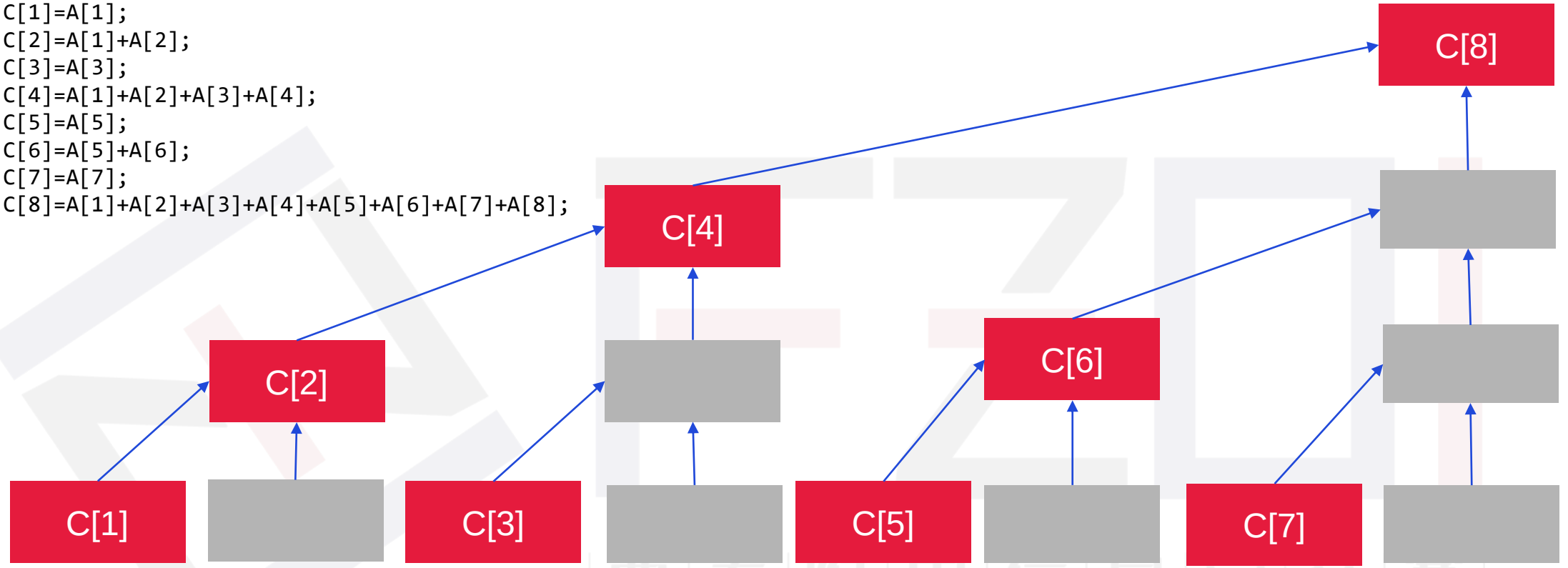
原数组

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

树状数组

西南大学附属中学
High School Affiliated to Southwest University

```
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];
```



原数组

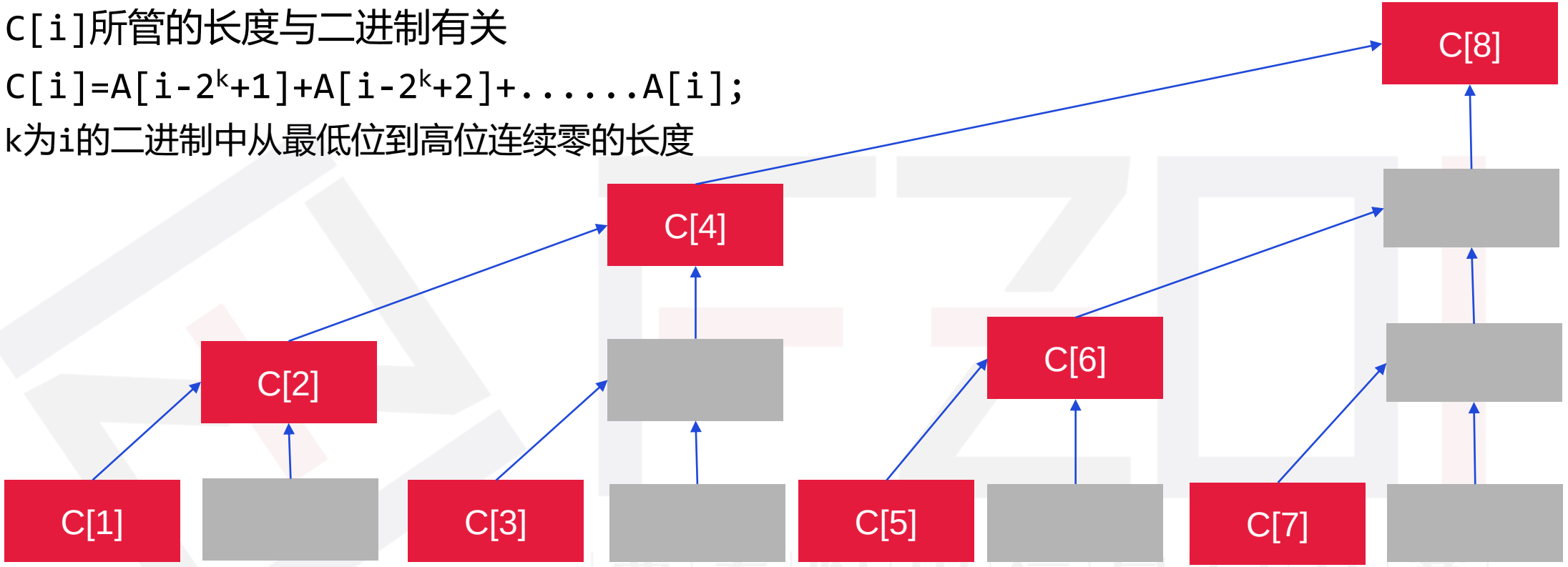
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

树状数组

C[i]所管的长度与二进制有关

C[i]=A[i-2^k+1]+A[i-2^k+2]+.....A[i];

k为i的二进制中从最低位到高位连续零的长度



原数组	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
下标二进制	0001	0010	0011	0100	0101	0110	0111	1000

树状数组-lowbit(x)

如何求解2^k？

引入lowbit(x) 函数
lowbit(x) 其实就是取出x的最低位1 ，换言之 **lowbit(x)=2^k**

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)

-6就是1010，-6&6 就是 1010 & 0110==0010，换算成十进制就是2，所以lowbit（6）==2

树状数组-lowbit(x)

如何求解2^k？

引入lowbit(x)
lowbit(x) 其实就是取出x的最低位1 换言之 **lowbit(x)=2^k**

```
int lowbit(int x){  //低位1
   return x&(-x);
}
```

C[i]=A[i-2^k+1]+A[i-2^k+2]+.....A[i];
C[i]=A[i-lowbit(i)+1]+A[i-lowbit(i)+2]+.....A[i];

操作1：区间查询

sum[7]=A[1]+A[2]+A[3]+A[4]+A[5]+A[6]+A[7] ;
前i项和
C[4]=A[1]+A[2]+A[3]+A[4];
C[6]=A[5]+A[6];
C[7]=A[7];

可以推出:　sum[7]=C[4]+C[6]+C[7];
序号写为二进制: sum[(111)]=C[(100)]+C[(110)]+C[(111)];

对于i=7 进行演示
7(111)
ans+=C[7]
lowbit(7)=001　7-lowbit(7)=6(110)　ans+=C[6]
lowbit(6)=010　6-lowbit(6)=4(100)　ans+=C[4]
lowbit(4)=100　4-lowbit(4)=0(000)

```
int getsum(int x){  //区间求和
        int ans=0;
        for(int i=x;i>0;i-=lowbit(i))
                ans+=C[i];
        return ans;
}
```

操作2：单点更新

当给A[1]加上一个y时，需要向上更新C[1] ,C[2],C[4],C[8]

```
  C[1]     C[2]     C[4],     C[8]
C[(001)] C[(010)] C[(100)] C[(1000)]

1(001)          C[1]+=y
lowbit(1)=001 1+lowbit(1)=2(010)    C[2]+=y
lowbit(2)=010 2+lowbit(2)=4(100)    C[4]+=y
lowbit(4)=100 4+lowbit(4)=8(1000)   C[8]+=y
```

```
void update(int x,int y){  // 单点更新
        for(int i=x;i<=n;i+=lowbit(i))
                c[i]+=y;
}
```

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

建树的过程就是对于每个位置i进行一次update

小结


```
int lowbit(int x){  //求低位1
    return x&(-x);
}
```

```
void update(int x,int y){  // 单点更新
        for(int i=x;i<=n;i+=lowbit(i))
                c[i]+=y;
}
```

```
int getsum(int x){  //区间求和
        int ans=0;
        for(int i=x;i>0;i-=lowbit(i))
                ans+=c[i];
        return ans;
}
```

树状数组能否处理下标0开始的情况?

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

应用：求逆序对

已知求逆序对的方法：归并排序
树状数组也可以求逆序对

例子：给定的序列为 4 3 2 1，我们从左往右依次将给定的序列输入，每次输入一个数a时，就将当前序列中大于a的元素的个数计算出来，并累加到ans中，最后ans就是这个序列的逆序数个数

序列的变化(下划线为新增加元素)	序列中大于新增加的数字的个数	操作
{ }	0	初始化时序列中一个数都没有
{4 }	0	往序列中增加4，统计此时序列中大于4的元素个数
{4 3 }	1	往序列中增加3，统计此时序列中大于3的元素个数
{4 3 2}	2	往序列中增加2，统计此时序列中大于2的元素个数
{4 3 2 1}	3	往序列中增加1，统计此时序列中大于1的元素个数

Q：如果是-1 1 100 1001 10000求逆序对怎么办？

离散化

应用：求逆序对

已知求逆序对的方法：归并排序
树状数组也可以求逆序对

```
int lowbit(int i){
    return i&(-i);
}
int insert(int i,int x){
    while(i<=n){
        c[i]+=x;
        i+=lowbit(i);
    }
    return 0;
}

int getsum(int i){
    int sum=0;
    while(i>0){
        sum+=c[i];
        i-=lowbit(i);
    }
    return sum;
}
void output()
{
    for(int i=1;i<=n;i++) cout<<c[i]<<" ";
    cout<<endl;
}
```

```
int main()
{
    while(cin>>n){
        int ans=0;
        memset(c,0,sizeof(c));
        for(int i=1;i<=n;i++){
            int a;
            cin>>a;
            insert(a,1);
            ans+=i-getsum(a);//统计当前序列中大于a的元素的个数
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

二维树状数组


```
void  update(int x,int y,int z)//(x,y)增加z
{
        int i=x;
        while(i<=N){
            int j=y;
            while(j<=N){
                c[i][j]+=z;
                j+=lowbit(j);
            }
            i+=lowbit(i);
        }
}
```

```
int  getsum(int x,int y){
        int ans=0,i=x;
        while(i>0){
            int j=y;
            while(j>0){
                ans+=c[i][j];
                j-=lowbit(j);
            }
            i-=lowbit(i);
        }
        return ans;
}
```

其他变式操作，请自学https://blog.csdn.net/bestsort/article/details/80796531

树状数组的兄弟：**线段树**

能用树状数组做的，线段树也能做

树状数组胜在实现容易，程序常数小

离散化

原数据：a[ ] :1，3，100，2000，500000；（有序）
映射到：0，1，2，3，4；（保序离散化，小的在前，大的在后，其实就是把a中的值映射到下标）

```
int main(){
            scanf("%d", &n);
            for(int i = 1; i < n+1; i++){
                        scanf("%d", a+i);
                        b[i] = a[i]; //复制一份
            }
            sort(b+1, b+1+n);
            int e = unique(b+1, b+1+n)-b-1; //去重,并得到去重后的个数
            for(int i = 1; i < n+1; i++)a[i] = lower_bound(b+1, b+1+e, a[i])-b;
            for(int i = 1; i < n+1; i++)printf("%d ", a[i]);

            return 0;
}
```

离散化步骤：

排序
去重
二分查找大小

Thanks

For Your Watching