## 概要

普通的前缀和方法在原数组值被修改后，更新时效率较低，使用树状数组可以将原数组有规律地分段，实现跳跃式的操作，查找与修改都较快。

lowbit指二进制数中最右边的1所在位置的权值，如1100的lowbit为4，使用(x&-x)可以快速求得lowbit。

树状数组下标从1开始，对应的树就是按下标组织起来的，每一个节点作为下标，对应的值代表原数组中一段元素的和。

树的第i层（层数向上递增，从1开始）是所有lowbit为i的下标，从左往右递增。假设下标为j，则节点对应的值为原数组自j下标起，向前的一段段长为2^(i-1)的元素的和。

左节点加上lowbit，右节点减去lowbit得到父节点，反过来得到父节点的父节点。

起初数组元素全置零，修改值使用add操作，向右上方寻找相关的段（即不断加lowbit直到超出下标范围）都加上该值，求前缀和使用sum操作，向左上方寻找相关的段（即不断减lowbit直到下标小于1）并加起来，正好得到原数组起始到该下标的所有元素的和。

## 固定长度的上升与下降子序列个数

给出一个元素各不相同的序列ab[] ，求长度为3的上升子序列和下降子序列个数。

以子序列中间值来计算，看它前面有多少个比自己小的元素，那剩下的就是比自己大的，右边同理，最后用乘法法则计算并求和。

问题转换为对于每个ab[i]，计算左边比它小的元素个数low[i]，右边比它小的元素个数high[i]，则以ab[i]为中间值的序列个数为：

low[i]\*(n-i-high[i])+(i-low[i]-1)\* high[i]

设一个辅助数组help[]，help[j]==1表示在目前扫描过的元素中存在等于j的元素 ，初始置零，先从左到右扫描ab[i]，置help[ab[i]]=1，low[i]即为help[ab[i]-1]前缀和。

因为序列中元素都是不同的，并且扫描是有先后顺序，很显然help[i]之前元素就记录了比i小的元素个数。

求high[i]则从右往左扫描，边扫描边更新并求前缀和。

## 坐标变换的树状数组

一共有n个光碟叠在一起，从上往下编号1到n，每次问编号为i的光碟的上面有多少光碟account[i]，然后将该光碟放在最上面。

可以发现每次放好后，i上面的光碟account都加1，i下面的光碟account都不变，i本身account变成0。

若k位置有光盘用1表示，则将i之前的1累加起来得到account[i]，可使用前缀和实现。考虑用树状数组求前缀和，每次将i上值置为-1，将i移到最前面，值置为1

这样如果正着存（顶部朝下标减小方向）要把数据一开始放在x[]右半部分（因为坐标会向左边扩展），若倒着存需一开始放左半部分。

正着存add时要加上多余的，倒着存add时要更新多余的，都麻烦。

不如就正着存。

另开数组存下标映射（光盘编号到数组下标的映射）。