**[树状数组 Review](http://roba.ycool.com/post.1467264.html)**

树状数组是一个可以很高效的进行区间统计的数据结构。在思想上类似于线段树，比线段树节省空间，编程复杂度比线段树低，但适用范围比线段树小。

以简单的求和为例。设原数组为a[1..N]，树状数组为c[1..N]，其中c[k] = a[k-(2^t)+1] + ... + a[k]。比如c[6] = c[5] + c[6]。也就是说，把k表示成二进制1\*\*\*10000，那么c[k]就是1\*\*\*00001 + 1\*\*\*00010 + ... + 1\*\*\*10000这一段数的和。设一个函数lowestbit(k)为取得k的最低非零位，容易发现，根据上面的表示方法，从a[1]到a[k]的所有数的总和即为sum[k] = c[k] + c[k-lowestbit(k)] + c[k-lowestbit(k)-lowestbit(k-lowestbit(k))] + ... 于是可以在logk的时间内求出sum[k]。当数组中某元素发生变化时，需要改动的c值是c[k],c[k+lowestbit(k)], c[k+lowestbit(k)+lowestbit(k+lowestbit(k))] ... 这个复杂度是logN (N为最大范围)

扩展到多维情况：以二维为例，用c[k1][k2]表示c[k1-(2^t1)+1][k2-(2^t2)+1] + ... + c[k1][k2]的总和。可以用类似的方法进行处理。复杂度为(logn)^k (k为维数)

树状数组相比线段树的优势：空间复杂度略低，编程复杂度低，容易扩展到多维情况。劣势：适用范围小，对可以进行的运算也有限制，比如每次要查询的是一个区间的最小值，似乎就没有很好的解决办法。

多维情况的几道题目:

POJ 2155 Matrix  
URAL 1470 UFOs

其中POJ 2155是一道很不错的题目，表面上看，这题的要求似乎和树状数组的使用方法恰好相反，改变的是一个区间，查询的反而是一个点。实际上可以通过一个转化巧妙的解决。

首先对于每个数A定义集合up(A)表示{A, A+lowestbit(A), A+lowestbit(A)+lowestbit(A+lowestbit(A))...} 定义集合down(A)表示{A, A-lowestbit(A), A-lowestbit(A)-lowestbit(A-lowestbit(A)) ... , 0}。可以发现对于任何A<B，up(A)和down(B)的交集有且仅有一个数。

于是对于这道题目来说，翻转一个区间[A,B]（为了便于讨论先把原问题降为一维的情况），我们可以把down(B)的所有元素的翻转次数+1，再把down(A-1)的所有元素的翻转次数-1。而每次查询一个元素C时，只需要统计up(C)的所有元素的翻转次数之和，即为C实际被翻转的次数。

实际实现时，由于只考虑奇偶，因此无须统计确切的翻转次数。另外，如果翻转up(A)和up(B+1)，查询down(C)，也是同样的效果。这种方法可以很容易地扩展到二维情况。比起线段树、四分树之类的常规思路，无论编程复杂度还是常数速度上都有很大优势。