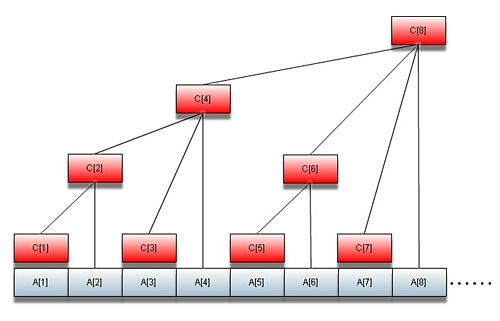
**树状数组总结**

树状数组，即Binary Index Tree，可以用来高效地维护动态数组a[1..n]的前缀和s[i]=a[1]+a[2]+...+a[i]。显然，对树状数组有两种基本的操作：改变a[i]的值，询问前缀和s[i]。树状数组通过维护一个数组c[1..n]来支持这两种操作，其中c[i]为a[i-2^k+1]到a[i]的和，k为i的二进制表示中末尾0的个数。我们可以用下面这个函数lowBit快速得到2^k（2的k次幂）的值：

inline int lowBit(int i) { return i & (-i); }

接下来我们看一张图，形象地理解一下树状数组：（转自吴豪's Blog）



如图所示，红色矩形表示的就是树状数组c[]。当改变a[i]的值的时候，我们沿着由a[i]（蓝色矩形）引出的线，一路上溯，更新所有相关的c[]值。当询问前缀和s[i]的时候，将s[i]分解为多个c[]值的和，就可以得到结果。经过深入的观察我们可以直接得到下面的两个函数：

void add(int i, int d) {

for (; i <= n; i += lowBit(i)) c[i] += d;

}

int sum(int i) {

int s = 0;

for (; i > 0; i -= lowBit(i)) s += c[i];

return s;

}

这里我们假定对a[i]的改变只是增加a[i]的值，实际上也可以是其他操作，实现上只要以原来的值为基准得到d即可。上面所述的均为一维树状数组，由此可以扩展至多维，下面给出二维树状数组的实现概要：

void add(int i, int j, int d) {

for (; i <= n; i += lowBit(i))

for (int k = j; k <= m; k += lowBit(k))

c[i][k] += d;

}

int sum(int i, int j) {

int s = 0;

for (; i > 0; i -= lowBit(i))

for (int k = j; k > 0; k -= lowBit(k))

s += c[i][k];

return s;

}

想更具体地理解树状数组还需要多做习题，POJ 1195就是一道比较基础的题目，可以直接用二维树状数组解决。POJ 2352,2481,3067这三道题本质上是一样的，都是对多个区间的一个端点有序化之后，用另一个端点作为索引，操作树状数组。POJ 1990在前面三道题的基础上更进一步，要用到两个一维树状数组（即按音量排序后，分别记录之前坐标更小的牛的个数，和这些坐标的总和）。POJ 3321也是对区间的操作，但难点在于如何将苹果树转化为区间。可以对树进行一次DFS并记录时间戳d[i]和f[i]，这样就将每棵子树限定在区间(d[i],f[i])里面了。要改变i的状态，就可以记录改变量到d[i]；要求子树的苹果数目，只需要知道子树的改变量即可，也就是sum(f[i])-sum(d[i]-1)。

做完上面这几道题，对树状数组就应该有一个比较深刻的理解了。接下来就可以去做POJ 2155，是楼教主出的一道题目。读过题后会发现正好跟我们树状数组的两个操作是反着的，回想一下做过的题目，好好体会一下for循环里索引变化的意义，不难得出下面的一维结构：

void add(int i, int d) {

for (; i > 0; i -= lowBit(i)) c[i] += d;

}

int sum(int i) {

int s = 0;

for (; i <= n; i += lowBit(i)) s += c[i];

return s;

}

理解好上面的一维结构，就可以很容易地写出Matrix这道题了。如果理解好了上面那张图，算法的正确性应该是不言而喻的。

树状数组在某些情况下可以用来替代传统的线段树，不仅可以降低编码的难度，同时在时空上的效率提升都很明显。而线段树可以解决的RMQ问题，其实也是可以用树状数组解决的，只不过要麻烦一些，每次操作的时间复杂度也不再是O(lg(n))，不过仍然是很高效，性价比也很高的一种解决方法。具体的原理和实现我会在RMQ问题的总结中写出来。