其实学树状数组说白了就是看那张图，那张树状数组和一般数组的关系的，看懂了基本就没问题了，推荐下面这个教程：
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees

下面对树状数组进行一些分析。

```
inline int Lowbit(int x)
{
    return x & (-x);
}

void Update(int x, int c)
{
    int i;
    for (i = x; i < maxn; i += Lowbit(i))
    {
        tree[i] += c;
    }
}

int Getsum(int x)
{
    int i;
    int temp(0);
    for (i = x; i >= 1; i -= Lowbit(i))
    {
        temp += tree[i];
    }
    return temp;
}
```

以上三个函数可以说是树状数组的“看家本事”，树状数组的高效就体现在这三个函数上了。我们现在对三个函数进行下分析。

Lowbit(x)，是求出 2^p(其中 p 为 x 的二进制表示中最右边的那个 1 的位置)，如 6 的二进制表示为 110，最右边的 1 为 1，故 Lowbit(6) = 2^1 = 2。

Update(x, c)，是使 x 这点的值改变 c，如果是一般数组改变的就是 x 自己这点，但是树状数组中要把(x, x+Lowbit(x), x+Lowbit(x+Lowbit(x)),…)这条路径的点都要改变 c，这样做是为了后面能够高效地求和。

Getsum(x)，是求的(1, …x-Lowbit(x-Lowbit(x))), x-Lowbit(x), x)这条路径的点的和，换句话说就相当于求一般数组 a[1]到 a[x]的和。

树状数组的高效就在于：与一般数组不同，一般数组都是下标不断加一来遍历的，而树状数组是不断加 2^p 来变化的，故效率为(logn)级别的。

树状数组的最基本功能就是求比某点 x 小的点的个数(这里的比较是抽象的概念，可以使数的大小，坐标的大小，质量的大小等)。

比如给定个数组 a[5] = {2, 5, 3, 4, 1}，求 b[i] = 位置 i 左边小于等于 a[i]的数的个数.如 b[5] = {0, 1, 1, 2, 0}，这是最正统的树状数组的应用，直接遍历遍数组，每个位置先求出 Getsum(a[i])，然后再修改树状数组 Update(a[i], 1)即可。当数的范围比较大时需要进行离散化，即先排个序，再重新编号。如 a[] = {10000000, 10, 2000, 20, 300}，那么离散化后 a[] = {5, 1, 4, 2, 3}。

但我们想个问题，如果要求 b[i] = 位置 i 左边大于等于 a[i]的数的个数呢？当然我们可以离散化时倒过来编号，但有没有更直接的方法呢?答案是有。几乎所有教程上树状数组的三个函数都是那 样写的，但我们可以想想问啥修改就是 x 不断增加，求和就是 x 不断减少，我们是否可以反过来呢，答案是肯定的。

```
void Update(int x, int c)
{
    int i;
    for (i = x; i >= 1; i -= Lowbit(i))
    {
        tree[i] += c;
    }
}

int Getsum(int x)
{
    int i;
    int temp(0);
    for (i = x; i < maxn; i += Lowbit(i))
    {
        temp += tree[i];
    }
    return temp;
}
```

我们只是将两个函数中的循环语句调换了下，现在每次要修改点 x 的值，就要修改(1, …x-Lowbit(x-Lowbit(x))), x-Lowbit(x), x)路径，而求和就变成求(x, x+Lowbit(x), x+Lowbit(x+Lowbit(x))),…)这条路径的点。而这不正好就是大于等于 x 的点的求和吗？

所以我们既可以修改 x 增大的路，求和 x 减小的路；也可以修改 x 减小的路，求和 x 增大的路，根据题目的需要来决定用哪种。

如果你已经掌握了上述的方法，那么基本可以解决了大部分树状数组的问题了～～

为什么说是大部分的问题呢，接下来看到的这个例子将会颠覆你前面建立起来的理念。

POJ 2155
http://acm.pku.edu.cn/JudgeOnline/problem?id=2155
楼教主出的题目～～原题是二维的，我们先化简为一维的讨论。

题目要求有两种操作 ：
1. 改变某个(a, b)内的所有。
2. 求某个点(a)的值。

这题网上的解题版本几乎都是直接抄百度百科中的树状数组的讲解
http://baike.baidu.com/view/1420784.htm(这个版本我理解了好长时间...)

以下是个人的理解：

很明显，这题与树状数组的操作正好相反。我们可以想想树状数组中的两个函数 Update()修改某个点的值(准确说是某个路径，递增或递 减)，Getsum()求和区间(1, x)内的点的值。我们上面已经分析了，这两个函数本质上是相同的，可以互相调换的。换句话说就是修改某点 x 和求和某个区间(1, x)是可以互相调换的，即我们可以用 Getsum(x, c)修改(1, x)这个区间内的点的值，而用 Update(x)来求该点的值。而两个函数的写法与原来完全相同(或者说就函数名调换了下)，仅仅是思想变化了下。(这里 很难理解就是因为只是思想变化，而程序与原来基本没变)

那么回到原题，我们要使区间(a, b)内的点 + c，只需要使区间(1, b)内的点+c，而区间(1, a-1)内的点-c 即可。如果是二维的，修改矩阵(x1, y1)到(x2, y2)，即(x2, y2)+c, (x1-1,y2)-c, (x2, y1-1)-c, (x1-1,y1-1)+c 即可。

总结：通过以上的分析，我们可以发现其实 Update()和 Getsum()这两个函数是相同的，我们可以用 Up()和 Down()来代替它们。Up()为操作 x 递增的路径，Down()为操作 x 递减的路径。

Up()和 Down() 有四种组合 ：

1. Up()表示修改单点的值，Down()表示求区间和。
2. Down()表示修改单点的值，Up()表示求区间和。
3. Up()表示修改区间，Down()表示求单点的值。
4. Down()表示修改区间，Up()表示求单点的值。

1 和 2 根据求比它大还是比它小来选择，而 3 和 4 种适用条件则相同，用其中一种即可。

以上是本人对于树状数组的一些理解，请各位大牛指点～～
下一篇文章将会对树状数组在OJ中出现的题目进行汇总(链接:

http://hi.baidu.com/czyuan%5Facm/blog/item/af6fe8a9177f7ef51e17a2ea.html)～～

czyuan 原创，转载请注明出处。

上一篇我们对树状数组进行了一些分析(详见树状数组学习系列 1 之 初步分析——czyuan原创 http://hi.baidu.com/czyuan_acm/blog/item/49f02acb487f06f452664fbc.html），这篇主要是对各大OJ有关树状数组的题目进行汇总。

先提个注意点，由于 Lowbit(0) = 0，这会导致 x 递增的那条路径发生死循环，所有当树状数组中可能出现 0 时，我们都全部加一，这样可以避免 0 带来的麻烦～～

简单：

POJ 2299 Ultra-QuickSort
http://acm.pku.edu.cn/JudgeOnline/problem?id=2299
求逆序数，可以用经典的归并排序做，也是基本的树状数组题目。

POJ 2352 Stars
http://acm.pku.edu.cn/JudgeOnline/problem?id=2352
题目意思就是求每个星星左下方的星星的个数，由于y轴已经排序好了，我们可以直接用按x轴建立一维树状数组，然后求相当于它前面比它小的个数，模板直接一套就搞定了～～

POJ 1195 Mobile phones
http://acm.pku.edu.cn/JudgeOnline/problem?id=1195
二维的树状数组，直接把Update()和Getsum()改为二维即可。
如Update()函数改为：

```
void Update(int x, int y, int d)
{ // 注意：当i = 0, 0 + Lowbit(0) = 0，会造成死循环!
    int i, j;
    for (i = x; i < maxn; i += Lowbit(i)) // 注意这里是maxn，是tree[]的大小.
    {
        for (j = y; j < maxn; j += Lowbit(j))
        {
            tree[i][j] += d;
        }
    }
}
```

POJ 2481 Cows

http://acm.pku.edu.cn/JudgeOnline/problem?id=2481

将E从打到小排序，如果E相等按S排序，然后就跟POJ 2352 Stars做法一样了～～

POJ 3067 Japan

http://acm.pku.edu.cn/JudgeOnline/problem?id=3067

先按第一个坐标排序从大到小排序，如果相等按第二个坐标从大到小排序，然后就又是跟Cows和Stars做法相同了...

POJ 2029 Get Many Persimmon Trees

http://acm.pku.edu.cn/JudgeOnline/problem?id=2029

O(n ^ 2)枚举起点，再用二维树状数组求其中的点数即可。

HOJ 2275 Number sequence

http://202.118.224.210/judge/show.php?Contestid=0&Proid=2275

两个一维树状数组，分别记录在它左边比它小的和在它右边比它大的即可～～

HOJ 1867 经理的烦恼

http://202.118.224.210/judge/show.php?Contestid=0&Proid=1867

先筛法求素数，然后如果从非素数改变成素数就Update(x, 1)，如果从素数改变成非素数就Update(x, -1)即可。

Sgu 180 Inversions

http://acm.sgu.ru/problem.php?contest=0&problem=180

经典树状数组 + 离散化。注意结果要用long long～～

SPOJ 1029 Matrix Summation

https://www.spoj.pl/problems/MATSUM/

基本的二维树状数组...

中等：

POJ 2155 Matrix

http://acm.pku.edu.cn/JudgeOnline/problem?id=2155

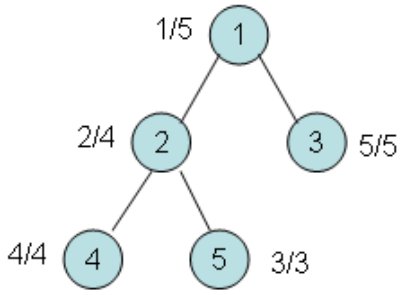
经典树状数组题目，分析见前一篇文章（树状数组学习系列 1 之 初步分析——czyuan原创）～～

POJ 3321 Apple Tree

http://acm.pku.edu.cn/JudgeOnline/problem?id=3321

这题的难点不在于树状数组，而是如果将整棵树映射到数组中。我们可以用DFS()改时间戳的方法，用begin[i]表示以i为根的子树遍历的第一个点，end[i]表示以i为根的子树遍历的最后一个点。

比如数据为：



5
1 2
2 5
2 4
1 3

那么begin[] = {1, 2, 5, 4, 3}, end[] = {5, 4, 5, 4, 3}，下标从1开始。

对于每个点都对应一个区间(begin[i], end[i])，如果要改变点a的状态，只要Update(begin[a])，要求该子树的苹果树，即Getsum(begin[a]) - Getsum(end[a] + 1)，(注：这里求和是求x递增的路径的和。)

POJ 1990 MooFest

http://acm.pku.edu.cn/JudgeOnline/problem?id=1990

这题的难点是要用两个一维的树状数组，分别记录在它前面横坐标比它小的牛的个数，和在它前面横坐标比它小的牛的横坐标之和。

按音量排个序，那么式子为：

ans += 1LL * cow[i].volumn * (count * x - pre + total - pre - (i - count) * x);

cow[i].volumn为该牛的能够听到的音量。

count为在第i只牛前面横坐标比它小的牛的个数。

pre为在第i只牛前面横坐标比它小的牛的横坐标之和。

total 表示前i - 1 个点的x坐标之和。

分为横坐标比它小和横坐标比它大的两部分计算即可。

Hdu 3015 Disharmony Trees

http://acm.hdu.edu.cn/showproblem.php?pid=3015

跟上题方法相同，只要按它的要求离散化后，按高度降序排序，套用上题二个树状数组的方法即可。

HOJ 2430 Counting the algorithms

http://202.118.224.210/judge/show.php?Contestid=0&Proid=2430

这题其实是个贪心，从左往右或者从右往左，找与它相同的删去即可。先扫描一遍记录第一次出现和第二次出现的位置，然后我们从右到佐，每删去一对，只需要更改左边的位置的树状数组即可，因为右边的不会再用到了。

tju 3243 Blocked Road

http://acm.tju.edu.cn/toj/showp3243.html

这题主要在于如果判断是否连通，我们可以先用j = Getsum(b) - Getsum(a - 1)，如果j等于(b - a)或者Getsum(n) - j等于(n - (b - a))，那么点a, b联通。

SPOJ 227 Ordering the Soldiers

http://www.spoj.pl/problems/ORDERS/

这题与正常的树状数组题目正好想反，给定数组b[i]表示i前面比a[i]小的点的

个数，求a[]数组。
我们可以先想想朴素的做法，比如b[] = {0, 1, 2, 0, 1}，我们用数组c[i]表示还存在的小于等于i的个数，一开始c[] = {1, 2, 3, 4, 5}，下标从 1 开始。

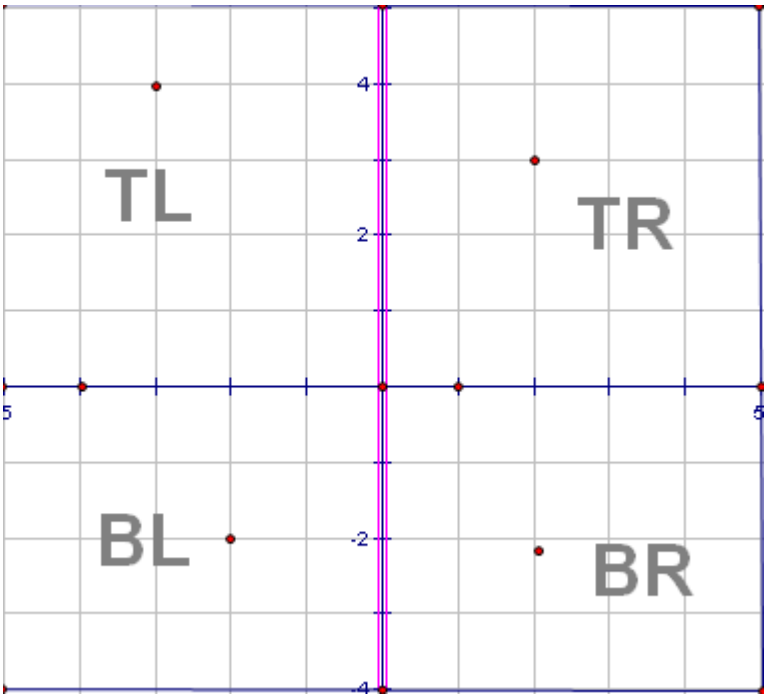
我们从右向左扫描b[]数组，b[5] = 1，说明该点的数是剩下的数中第 4 大的，也就是小于等于它的有 4 个，即我们要找最小的j符合c[j] = 4(这里可以想想为什么是最小的，不是最大的，挺好理解的)，而c[]是有序的，所以可以用二分来找j，复杂度为O(logn)，但现在问题是每次更新 c[]要O(n)的复杂度，这里我们就想到树状数组，c[i]表示还存在的小于等于i的个数，这不正好是树状数组的看家本领吗～～所以处理每个位置的复杂 度为O(logn * logn)，总的复杂度为O(n * logn * logn)。

hdu 2852 KiKi's K-Number
http://acm.hdu.edu.cn/showproblem.php?pid=2852

这题与上面那题类似，只是要求比a大的第k大的数，那我们用Getsum(a)求出小于等于a的个数，那么就是要我们求第k + Getsum(a)大的数，而删除操作只要判断Getsum(a) - Getsum(a - 1)是否为 0，为 0 则说明a不存在。

难题：

POJ 2464 Brownie Points II
http://acm.pku.edu.cn/JudgeOnline/problem?id=2464



这道题用二分也可以做的，这里介绍下树状数组的做法。首先有n个点，过每个点可以做x,y轴，把平面切成BL, TL, TR, BR四个部分，我们现在的问题是如果快速的计算这四个部分的点的个数。

这样我们可以先预处理，先按y坐标排序，求出每个点正左方和正右方的

点的个数LeftPoint[], RightPoint[]，复杂度为O(n)，同样我们再以x坐标排序，求出每个点正上方和正下方点的个数UpPoint[], DownPoint[]。还要求出比点i y坐标大的点的个数 LageY[]。注意：这里要进行下标映射，因为两次排序点的下标是不相同的。

然后按x坐标从小到大排序，x坐标相等则y坐标从小到大排序。我们可以把y坐标放在一个树状数组中。

对于第i个点，求出Getsum(y[i])即为BL的个数，然后Update(y[i])。由于现在是第i点，说明前面有i - 1 个点， 那么

TL = i - 1 - LeftPoint[i]- BL;

TR = LargeY[i] - TL - UpPoint[i] ;

BR = n - BL - TL - TR - LeftPoint[i] - RightPont[i] - UpPoint[i] - DownPont[i] - 1;

这样我们就求出四个部分的点的个数，然后判断有没有当前解优，有的话就更新即可～～

UVA 11610 Reverse Prime

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&category=78&problem=2657&mosmsg=Submission+received+with+ID+7313177

一道很综合的树状数组题，用到了树状数组中的很多知识点，包括离散化，二分查找等。

1. 先按题目要求筛法素数，找到所有的Reverse Prime。

2. 将这些Reverse Prime离散化，只有 78500 个左右。树状数组中tree[i]记录比i小的点的个数。

当执行q a操作时，二分查找最小的j, 使得Getsum(j) 等于 ++a(因为a可能为 0，所以统一加一)。这步与上面SPOJ 227 Ordering the Soldiers 类似。

3. 当执行d a操作时，先找到a离散化后的值b，然后Update(b, -1)即可。

按这样做后，运行时间为: 0.3s多，感觉很诧异，因为都是 0.1s以下的。这里特别感谢liuzhe大牛的指点，其实题目中的Reverse Prime是由 10^6 以下的素数倒置得到的，那么得到的要求是 7 位，最后一位一定是 0，我们可以对每个除以 10 处理，那么数的范围就减小了 10 倍，速度 就提高了不少。

最后自己又加了点优化，跑了 0.056s，排在第 3 名～～

3 7313367 czyuan 0.056 C++

树状数组题目总结

2007-08-16 20:15

hoj 1867 经理的烦恼

此题是最基本的一维树状数组题目，直接进行简单的加一减一（通过判素）操作即可。

hoj 2430 Counting the algorithms

从后往前不断删除（这样的话不存在区间包含问题），统计相同元素区间内数的个数。之后将这两个元素一起删除。

hoj 1640 Mobile Phone

该题目是典型的二维树状数组的题目。

hoj 2275 Number sequence

题目就是统计序列中 Ai < Aj > Ak（i < j < k）的个数，可以从前往后统计每个元素之前小于它的数的个数，在从后往前统计每个元素之后小于它的数的个数。然后乘积加和即可。注意树状数组统计是起始为 1。

poj 2352 Stars

相当经典的树状数组题目，一开始分析题目是第一感觉是二维的树状数组，不过数据范围显然不容许的，可先排序，然后再统计每个位置之前的星星的个数。

poj 3067 Japan

与 stars 极其相似，唯一的不同是上题诗统计之前的个数，而这个统计之后的个数，当然我们可以用当前总数 i 减去之前的数即可得到。此题 bt 之处在于不能用 long long 只能用__int64。

poj 2155 Matrix

此题是二维树状数组基本应用的变通，不像 hoj1640 简单的插入统计，而此题是对一个二维区间数值翻转（0 变 1，1 变 0），最后询问某处的值。所以可以使用奇偶性判断，这样我们可以使用树状数组在这个二维区间内加，就相当于翻转，最后统计某点翻转的次数。

poj 3321 Apple Tree

首先进行 DFS 遍历一遍，对每个节点编号（进入递归时 begin[i],出递归时 end[i]，具体见《算法导论》），由于子树的编号是连在一起的，同时包含在子树的的根 begin[root]和 end[root]之间，所以以后可以直接利用树状数组对其内容修改和统计。

hoj 2098 poj 2299 Ultra-QuickSort

该题是求逆序数的题目，当然我们可以使用合并排序法，在此介绍一种用树状数组解决的方法，由于元素的范围巨大，可以先采用离散化（就是排序编号），然后依次查值，并统计当前比自己大的元素的个数（当前总数-小于等于自己），即可得到结果，和以上的 Japan 有着相似的处理方法。