**树状数组学习系列1 之 初步分析——czyuan原创**

2009年08月13日 星期四 15:41

|  |
| --- |
| 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](http://hi.baidu.com/czyuan_acm/blog/item/af6fe8a9177f7ef51e17a2ea.html))～～  czyuan原创，转载请注明出处。 |