**树状数组总结**

2009-09-17 12:26

|  |
| --- |
| 树状数组是一种更新和查找都是log(N)的一种数据结构，  前两天做了一些这方面的题，总结了一下树状数组的基本几种形式。  首先还是lowbit这个函数，是树状数组的基本手段， int lowbit(int x) { return x&(-x); } 然后还是树状数组的基本概念，树状存储方式 C1 = A1 C2 = A1 + A2 C3 = A3 C4 = A1 + A2 + A3 + A4 C5 = A5 C6 = A5 + A6 C7 = A7 C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8  第一类树状数组，描述的是更新某位的情况，求解统计的情况， 比如： 1求解一串数中在第k前面比这个数小的个数， 2在一个一维的线上，若干个端点种有若干个权值，求解一条线上的所有端点权值之和。 第一个问题可以转换成第二类问题，就是数轴上预处理每个数的权值为0，每出现一次就权值＋1，然后统计从1到n的所有端点权值之和。 总的来说就是给出个例信息，求统计信息， 主要是两个函数 modify() 和 getsum() 一般都是 void modify(int i,int m)//i是需要更新的端点位置，m为更新的权值， {        while(i<=LIMIT)        {             a[i]+=m;             i+=lowBit(i);         } } int getSum(int i) {       int sum=0;        while(i>0)       {             sum+=a[i];            i-=lowBit(i);        }  } 解释一下,前面的第三行到第十一行表示的是树状数组存储的基本形式，当更新一个数时，比如更新5这个点，每次增加1，就是前面的所有等号右边出现5的式子中的C值都增加1，增加的过程是这样的，5二进制表示是101，首先C5＋＋，5加上lowbit()，变成110，于是C6++，再加上lowbit（）后，C8＋＋，以此类推，直至更新到LIMIT为止， 每个Ci表示的值就是Ci等式右边的权值之和，比如C8是A1到A8的所有和，C7只是A7的值， 在getsum（）的时候，比如查找1到6的所有值的和，6的二进制表示是110，首先累加C6，然后6减去lowbit（），变成100，然后累加C4，最后就能得到A1+A2+A3+A4+A5+A6了，这就是树状数组的基本存储形式决定的。  求解范围就是告知点的情况，求解线段的情况。  第二类树状数组，描述的是更新某段的情况，求解单个端点的情况， 大致可以转换为如下的问题： 在一条一维的土地，从0点到B点间的所有坑都种几粒种子，最后求解某个坑的权值情况。 这时候两个函数 modify（）和getvalue（）需要做一些变化 void modify(int i,int add) { while(i>0) {    a[i]+=add;    i-=lowbit(i); } } void getvalue（int i） { int res=0; while(i<=LIMIT) {    res+=a[i];    i+=lowbit(i); } return res; }  解释一下，这里跟第一类的两个函数只有微小的差别就是while里面循环的顺序掉了一下，但是里面的意思需要深刻的理解，还是以以下为例。 C1 = A1 C2 = A1 + A2 C3 = A3 C4 = A1 + A2 + A3 + A4 C5 = A5 C6 = A5 + A6 C7 = A7 C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 当更新1到7时，就是1到7的所有点都加1，7的进制是111，首先是C7＋＋，然后是C6++,然后是C4++，当getvalue的时候，统计的是所有含有A7的项Ci的和，含有A7的项就是111，C7，7加上lowbit（），1000 C8，和10000 C16，以及100000 C32，……累加所有含有该项的Ci，就能得到某个点更新的统计值了。  第二类求解的范围主要是告知段的情况，求解点的情况。  扩展：二维情况， 经典题目，二维的M\*N土地上种花生，告知若干个点的，求解一个矩形内花生总数之和。 方法1：使用min（m,n）次的一维树状数组，将二维的硬生生地给成几个一维的和。 复杂度O（T\*m\*log（n）） 方法2：使用二维的树状数组，复杂度O(T\*log(N)\*log(m))， 类似于一维的情况， C11 = A11 C21 = A11 + A21 C31 = A31 C12 = A12 + A11 C22 = A11 + A21 + A12 + A22  C32 = A31 + A32 C31 = A31 C32 = A31 + A32 C33 = A33 以上是3\*3的情况，可以看出Cij的两个下标的规律，分别对i,j进行lowbit操作即可， modify()和getsum() 函数 void modify(int x,int y,int in) { int j=y; while(x<=n) {    y=j;    while(y<=n)    {     a[x][y]+=in;     y+=lowbit(y);    }    x+=lowbit(x); } } int getsum(int x,int y) { int res=0; int j=y; while(x>0) {    y=j;    while(y>0)    {     res+=a[x][y];     y-=lowbit(y);    }    x-=lowbit(x); } return res; }  这是第一类树状数组扩展到二维的情况，第二类树状数组扩展到二维的情况也是如此。 注意点就是两次while的嵌套，以及y值的恢复。  二维的如此，三维的就是同样的道理，这里不一一赘述了。  第三类树状数组，混合第一类和第二类树状数组，第一类是已知点的情况，求线的情况，第二类是已知线的情况，求点的情况，第三类可以归结起来，就是即给点的情况，又给面的情况，既求解点的情况，又求解线的情况。 同样的第三类树状数组也可以扩展到二维，三维的情况，这里只说明下一维的情况， 先可以将问题化解一下，就是求解点i的情况，可以看成是线i-线(i-1)的值，算是半个容斥吧。 不过问题还是比较复杂，主要就是虽然前两类树状数组尽管形式上差不多，但记录存储的方式不一样，就是a[]数组里面的值是不一样的，虽然可以通过一一统计的方式将第三类树状数组转换为第一类和第二类，但是我们希望的是找到一种在log(n)之内能够完成modify和sum操作的方法。 提示一下，仔细观察树状数组的存储结构: C1 = A1 C2 = A1 + A2 C3 = A3 C4 = A1 + A2 + A3 + A4 C5 = A5 C6 = A5 + A6 C7 = A7 C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 在第一类中Ci表示Ci等式右边所有点的权值之和， 在第二类总Ci表示Ci等式右边每个点的权值。  注意点，树状数组只能处理正整数，遇到负数或者0，或者小数，全部加上一个大常数，或全部乘以一个大数，使全部数据转换成正整数，时间复杂度每次插入和统计的复杂度都是log N ，如果更新次数是T，则总得时间复杂度为O(T \*log N) 空间复杂度是O（N），N是所处理到的最大正整数LIMIT，一般最多只能为几十万的数量级别。  在以后遇到的题目中，不太可能将树状数组作为单独的题目算法，作为一种数据结构，可能仅仅是作为数据的存储。  还记得上次宁波赛区那个约瑟夫变种，需要树状数组加二分，使得复杂度控制在可以接受的范围内。 |