线段树在一类分治问题上的应用

杭州学军中学 徐寅展

摘 要

线段树与按时间分治都是信息学竞赛中常用的算法。本文提出利用线段树解决分治问题的思想，给出一个能将区间修改问题转化为分治问题的方法，并将这种思想推广到了简单的可持久化问题中，最后本文介绍了虚树的概念以及构造方法。本文的例题比较基础，希望能起到抛砖引玉的作用。

1 线段树与按时间分治

1.1 线段树

线段树是一种很常用的数据结构。它的根表示的区间为总区间，每一个非叶子节点都有一个左儿子与一个右儿子。如果这个节点表示的区间为$[l,r]$，那么它的左儿子表示的区间为$[l,\lfloor\frac{l+r}{2}\rfloor]$，右儿子表示的区间为$[\lfloor\frac{l+r}{2}\rfloor+1,r]$。

找到一段区间在线段树中的对应位置的时间复杂度是$O(\log n)$的，这里n是总区间的大小。

简单证明：令插入的区间为$[a,b]$，子树i表示的区间为$[l_i,r_i]$，$mid_i=\lfloor\frac{l_i+r_i}{2}\rfloor$。若在某次访问子树$i$时需要同时访问左右两棵子树，那么有$a\le mid_i$以及$mid_i<b$。在$i$的左子树中，若又有一个j使得$a\le mid_j$以及$mid_j<b$，则由$a\le mid_j$以及$r_j\le mid_i<b$，得$[a,b]$肯定完全覆盖$j$的右子树。$i$的右子树中情况类似。于是同时访问两棵子树的次数$\le 1$，每一层的访问次数都是$O(1)$的，总复杂度为$O(\log n)$。

1.2 按时间分治

按时间分治是一种分治方法。它的基本思想是若要处理时间$[l,r]$上的修改与询问，就先处理$[l,\lfloor\frac{l+r}{2}\rfloor]$上的修改对$[\lfloor\frac{l+r}{2}\rfloor+1,r]$上的询问的贡献，之后递归处

理$[l,\lfloor\frac{l+r}{2}\rfloor]$与$[\lfloor\frac{l+r}{2}\rfloor+1,r]$。这种方法要求各个修改对询问的贡献是**互不影响**的，修改是**互相独立**的，且题目还需**允许离线**。若处理k个修改与询问的时间复杂度为$O(f(k))$，那么由主定理易得总时间复杂度为$O(f(n)\log n)$。

1.3 有撤销操作的按时间分治问题

由于对时间分治要求操作是互相独立的，因此如果有形如“撤销某次操作”这样的操作，就不能应用按时间分治的方法。遇到这种情况时，需要采用一些方法来将原问题转化为没有删除操作的新问题。一个优秀的做法是进行一次分治后将时间倒过来，那么删除操作也就变为了原来的加入操作。关于这种**“时光倒流”**的方法可以参考许昊然2013年国家集训队论文。

本文将提出另外一种通用方法，可以解决有撤销操作、修改对询问的贡献互不影响，且允许离线的问题。这也是本文的主题。下面由一道例题来具体阐述这种方法。

【例一】动态半平面交问题[1]

给定一个平面，要求支持以下几种操作：

1.插入一个半平面

2.删除一个还在当前平面的半平面

3.询问某个点是否在当前半平面的交集内

如果没有第二个的操作，那么这就是一道很好的按时间分治练习题。

通过观察上面两节中线段树的定义与按时间分治的方法，可以发现它们有着一些联系：**每次都是从区间$[l,r]$变为[l,mid]和[mid+1,r]进行递归。**

先考虑没有删除操作的子问题。在对时间分治中，每次都处理在左区间中的修改，对在右区间中的询问的贡献，这在线段树中就相当于处理在左孩子中修改对在右孩子中询问的贡献。在分治的最后，剩下的区间是大小为1的，而这也就是把对应的操作插入到了线段树的叶子中。

我们惊讶地发现，**按时间分治等价于线段树**，整个过程就相当于建立了以时间为序的线段树！当然分治与线段树还是有区别的：一个是先处理大区间再处理小区间，一个是先处理小区间，再处理大区间。这时就有一个奇妙的想法：

[1]题目来源：经典问题

如果按线段树的处理顺序来解决分治问题，会不会有一些强大或者方便的方法呢?

具体的做法是，事先建立一棵**大小为操作个数**的线段树，依次把每一个操作插入到线段树中对应的叶子，这个叶子的位置即为这个操作对应的时间。考虑线段树中的一个非叶子节点x，把它左儿子中所有半平面拿出来，求一个上凸壳与一个下凸壳；对于它右子树中的询问点(X,Y)，依次判断每一个是否在上凸壳或者下凸壳中。判断的方法是二分找到在$x=X$上最高（或者最低）的半平面，与Y比较即可。当这个点有一次不在某个凸壳中，就把它标记为永久不在半平面的交集中。这个做法是$O(m\log^2 m)$的，这里m是操作总数。如果用归并将半平面的斜率以及询问点的x坐标排序，可以做到$O(m\log m)$。

问题是，上述做法似乎与直接分治相差无几。换个角度思考，考虑每一个半平面，它对哪些询问点有贡献呢? 当然是他之后的那些嘛。既然知道这一点，何不在线段树中**直接对这个区间打上一个标记**，表示这个区间中的点需要被这个半平面包含。

现在的算法流程变成了：若某个半平面出现的时间为T，那么在线段树中$[T,m]$对应的那些区间标记上这个操作。只要对线段树中的每一个节点，将标记它的那些半平面与它所对应的区间的询问点做一次半平面交与询问即可。

令人欣喜，这样的算法是可以推广到有第二个操作的问题的——**一个半平面依然对应于一段连续的时间**，所以剩下的工作就与上述算法大同小异了。

接下去考虑复杂度。不考虑排序，求一次半平面交，以及判断一些点是否在当前半平面交内是线性的。询问点的排序可以利用归并完成。半平面可以在插入线段树之前事先排序，**先插的半平面一定在后插的半平面之前**，因此每一个节点处理的时候那些半平面都是已经排序好了的。时空复杂度都是$O(m\log m)$。

上述做法的空间复杂度是比经典做法要劣的，怎样才能优化它呢? 问题的关键在于求出线段覆盖的节点后就**直接对这个节点代表的半平面做半平面交**。显然，可以直接把所有半平面一起插入线段树，按线段树中插入区间的方式递归这些半平面；同时，必须把询问点与半平面同时插入，否则空间得不到优化。但是直接这样做，空间复杂度还是不变的，因为若所有半平面都覆盖了某一个叶子，那么从这个叶子到根的路径都要存一遍所有半平面。虽然这种情况是不可能的，但是**依然可以构造出类似的较差的情况**。

似乎我们陷入了困境? 别忘了我们依然是在线段树上讨论。每一个半平面

插入的时候**只会分叉一次**，这同时也说明了线段树的每一层最多只会有4个同样的半平面。因此只要一层一层地执行算法，就能保证空间复杂度了。直观上理解，这就是由bfs取代了上段中的dfs。至此，我们得到了一个空间$O(m)$，时间$O(m\log m)$的优秀做法。

前两段的叙述表明，这里所述的线段树与分治的**本质是相同的，只是表现形式不同罢了**。分治每次只考虑了线段树中的一条路径，线段树即为整个分治过程。依靠线段树这个更加具体、直观的模型，可以更清晰地处理问题，更方便地思考出优化问题的方法。但分治依然具有其好写、快捷的优点。

【例二】作业[2]

给定一列n个数，每次询问在第l个数字到第r个数字中，权值在a到b中的数字个数，以及在这些数字中，不同的数字的个数，这样的询问有Q个。

$n \le 100000, Q \le 1000000$。

考虑到不同的数字与权值关系较大，因此对权值建立线段树，在权值不同的地方数字是不会相等的。权值线段树中的每一个点，存下的是权值在它所代表的范围内的那些数字的位置。每一个询问对应的权值是线段树中的一些节点。

在线段树的每一个节点中，问题就变为求$[l,r]$中的数字个数，以及求$[l,r]$中的不同数字个数。这两个问题都可以用树状数组做到$O((n+Q)\log n)$。

这种做法的空间复杂度是$O(n+Q)$，时间复杂度是$O((n+Q)\log^2 n)$。

1.4 一种通用的转化方法

由于在现今信息学竞赛中，有像“撤销某次添加的操作”这种操作的题目是比较少的，因此上文中的方法往往得不到很广泛的应用。下面给出一种方法，能将形如“把$[l,r]$当中的所有元素全部变为某一个相同的元素”这种操作，转化为添加与撤销操作。

具体做法如下：将修改操作依次插入到一棵按**位置**建的线段树中，并对$[l,r]$对应的那些节点添上标记。若在插入的过程中遇到了一个已经有标记的

[2]题目来源：Ahoi2013

节点，就把这个节点上的标记分别记到左右两个孩子上，并将这个节点上的标记清除。当一次修改操作在线段树上找到对应的节点之后，将那些节点的子树中的**所有标记**都取出来（并清除这些标记）。若当前修改操作的插入时间为i，某个取出的标记的插入时间为j，且这个标记在线段树中的位置为$[l,r]$，那么就在转化后的问题中增如下两个操作：

1.**在时间j将位置$[l,r]$上的元素全部改为某一个元素。**

2.**在时间i撤销1这个操作。**

实现时只要记下线段树**某个节点对应的子树中有无标记**。如果没有标记，则不必继续在这棵子树中寻找；否则，继续寻找那些标记。

简单审视一下这种做法的正确性与复杂度。

可以证明，在区间覆盖操作中，任意一个时刻，序列中的第i个元素即为线段树中第i个叶子到根的路径上的最后那个标记所代表的元素。而一个已标记节点所代表的子树中，**其他的标记是没有用的**，因此清除标记并不影响区间覆盖。每插入一个修改操作，都相当于把它对应的那个区间中的所有元素取出，将那些元素修改为这个元素（只不过把一些连续且相同的元素并起来了）。

接下来考虑复杂度。每次插入后，每一个有标记的节点的子树中是**没有其他标记**的。因此每插入到一个节点中时标记只会下传一次（而不会由于下传到的位置有标记而继续下传）。下传一次之后标记个数会增加$O(1)$个，标记总深度会增加$O(\log n)$。如果用m代表修改总数，n代表序列长度，总的插入复杂度是$O(m\log^2 n)$ 的，这包括了插入以及下传的总时间。清除复杂度是不会超过插入复杂度的，因此总的复杂度是$O(m\log^2 n)$ 的，新问题中的操作数不会超过总标记数$O(m\log n)$。

但是这样的复杂度显然不能满足我们的要求。尝试写一下代码后可以发现，虽然构造的时间是$O(m\log^2 n)$的，但是对于$n=m=200000$的数据，还是能在200ms以内出解的；输出操作次数之后，发现与$O(m\log n)$更为接近。

换个角度思考，在原序列中，每次新增一个区间修改操作，都会把一些完整的相同元素段覆盖，并把两端那两个相同元素段截成两半，且覆盖掉其中一半。**每次最多会增加两个新的相同元素段**，因此最后不同的元素段是$O(m)$的。如：将6覆盖到1112334445555的第3个位置到倒数第二个位置，会把“2”，“33”，“444”完全覆盖，“111”会分成“1”与“11”，“5555”会分裂成“55”与“55”。

回归线段树，每次清除的标记有许多都是对应于区间上连续的一段的，**只**

需要将这些标记并到一起，新问题中的总操作数就是$O(m)$的了；同时，查找出所有可以并到一起的标记，实际上是$O(\log n)$的，因为这相当于找到一段区间在线段树中对应的节点。所以构造复杂度为$O(m \log n)$。

【例三】一个简单的区间问题[3]

给定一个长度为n的序列，初始时每一个元素的值都是给定的。要求支持以下几种操作：

1.将序列中第l个到第r个元素全部修改为x。

2.询问序列中第l个到第r个元素中有多少元素的值小于等于x。

一共有m个操作。

为了简化叙述，下列表述中皆认为$m = O(n)$。

先利用上述方法转化，新问题中第一个操作变为在序列的第l个位置到第r个位置中，每一个位置增加一个元素x以及撤销某次增加操作。按照上一节中的方法，可以将问题变为只有区间增加某一个元素的操作，且所有增加操作在询问操作之前的子问题。这个子问题只要按权值大小排序以后，利用线段树维护区间加以及区间询问权值和就可以了。

考虑复杂度。修改操作起初是$O(n)$的，经过第一步转化后有$O(n)$个，再添加到时间线段树之后就有$O(n \log n)$个。这些操作每一个还要在线段树中修改，因此总的时间复杂度是$O(n \log^2 n)$的。考虑询问的部分，在时间线段树上每一个询问都要处理$O(\log n)$次，每一次要询问区间和，所以这部分的时间复杂度是$O(n \log^2 n)$的。总的时间复杂度为$O(n \log^2 n)$。

由于这题的特殊性，在第一步转化之后的第二个操作可以看作一个权值为-1的添加的操作，之后可以采取普通的按时间分治写法来减小常数，时间复杂度不变。

【例四】Robot[4]

有n个机器人，m个插座排成一排，开始时每一个插座都是空的。

每一个时刻，都会有一个机器人登上舞台。机器人i会选择插着插座的某个机器人j，在第k个插座处吸收它的能量（需要保证此时j号机器人插着k号插

[3]题目来源：经典问题

[4]题目来源：原创

头）。如果j机器人的能量为f_j，那么i号机器人的能量为$f_j+(i-j)^2+(i-k)^2+(j-k)^2$，吸收完之后$j$机器人的能量并不会消失。如果没有任何一个能吸收的机器人，i机器人的能量为0。

获得自己的能量之后，i机器人会把自己的插头插到每一个$[l,r]$的插座上（此时插着的插头会被全部拔掉）。

i机器人在$i+M$时刻会把自己的所有插头拔掉并离开。这个M对所有机器人都是相同的。

问每一个机器人能获得的最大能量是多少。

先不考虑能在$i+M$时刻拔掉插头。即使不能预先确定每一个f_i的值，每一个机器人的有效区间是确定的。按照上文中的方法处理过后，线段树中的每一个节点只有添加j号机器人插着一段插头的操作。按照线段树的**中序遍历处理**所有操作，便能得到所有f值了。

由于$f_i = max(f_j+(i-j)^2+(i-k)^2+(j-k)^2)$，在$i,j$确定时，$(i-k)^2+(j-k)^2$必然在端点处取到最大值，因此只需把每一个区间的两个端点取出来，中间那段是无用的。$f_j+(i-j)^2+(i-k)^2+(j-k)^2 = f_j+(j-k)^2+j^2+k^2-2(j+k)i+2i^2$。这个转移方程是可以**利用斜率优化**$O(n)$求解的。此时的做法时间复杂度是$O(n\log n+n\log m)$的。

拔掉插头的操作，这只需要找出p在$i+M$时刻，i号机器人的标记**当前存在了线段树的哪些节点当中**，将它们全部取出来即可，这与普通的撤销操作相同，只需要在时间线段树中打标记。时间复杂度依然是$O(n\log n+n\log m)$。

EXT

题目中其他所有条件都不变，只将最大能量改为最小能量。

改为最小能量后，只将每一个区间的两个端点取出来的做法是不对的，$f_j+(i-j)^2+(i-k)^2+(j-k)^2$的最小值还会在$k=\frac{i+j}{2}$时取到。这时必须有$l\le\frac{i+j}{2}\le r$，也就是$2l-j\le i\le 2r-j$。这可以在原来的基础上，**再进行一次类似的分治**。当然，由于k的定义域是整数范围的，还需要处理好边界情况，但这并不影响复杂度。时间复杂度为$O(n\log^2 n)$。

2 有关树形结构的分治

2.1 序列问题的可持久化

可持久化是各种数据结构的经典应用。这里只考虑“把当前全局变回到某一次操作之后的样子”这样的可持久化。

在每一个时间T，都会有若干次不同的修改操作将它变为新的分支T'，新的这些分支T'会有自己新的后代，但与其他分支中的状态无关。回到某次操作也就是回到某个节点罢了。因此最后修改操作形成的关系是一个树形结构。

每一个修改操作会影响它子树中的所有节点，在dfs一棵树时，这些子树中的节点被访问到的时间是连续的一段。因此，可以在刚进入子树x时，添加上x所代表的操作，离开子树x时再撤销x这个操作，撤销操作的撤销操作亦即一个添加操作。将这棵树中的节点按照dfs序排列之后，就将这个问题转化为了序列问题，且操作数量级不变。

2.2 虚树

在序列问题的分治中，每一个子问题的时间复杂度不能与总序列的长度有关，只能与这个子问题中包含的元素个数有关。这只需将这些元素按照在序列中的位置排序，就可以做到与序列总长无关了。

但由于树的结构比较复杂，不能简单地通过排序来摸清子问题中节点之间的关系，因此需要利用虚树来优化复杂度。

2.2.1 定义

设原树为一棵形态固定[5]的树T，它有n 个节点。它的虚树是它的一棵联通子树[6]，或者是它的某棵联通子树，将一些没有分叉的连续的边缩成一条边之后形成的树。由于这些新形成的边在T中是不存在的，因此形象地称之为虚树。

[5]形态不固定的树也能构造虚树，但与本文没有较大关系，这里不再介绍。感兴趣的读者可以查阅参考文献[3]。

[6]这里所说的子树概念类似与子图，而不是有根树的子树。

2.2.2 构造方法

在实际问题中，构造虚树的条件通常是求出一棵较简的虚树，使其包含$x_1, x_2, \ldots, x_m$这些节点，称这些节点为关键点。

首先，如果一个点是关键点，那么这个点肯定要是最后的虚树中的某个节点。将这些点之间路径上的所有点选出后形成的子树，就是一个合法的虚树（即使很不优）。考虑这些点中的某个非关键点，去掉这个点后整颗树会被分为许多子树，这些子树中至少有两棵包含关键点（否则这个点就不会被选）。如果这些子树中有两棵包含关键点，那么这个点连向这两个子树的那两条边就能缩为同一条边；否则，将这个点也作为最后虚树中的节点。

这样的构造还是有点麻烦的，可以降低一点要求。将整棵树视为有根树。若一个点满足上述的作为虚树节点的条件，那么被它分成的那些子树中，**至少有两个是它的儿子子树**；而它就对应于在那两棵子树中的**关键点的**Lca。因此只需将关键点两两之间的Lca作为虚树中的节点即可。

树上两个点之间的Lca，即这棵树的Euler tour technique中，那两个点之间深度最小的节点。如果有三个点x, y, z，它们在Euler tour technique 中对应的先后次序为x, y, z，由于区间最小值具有可合并性，那么有$h[Lca(x, z)] = min(h[Lca(x, y)], h[Lca(y, z)])$，而在这一段中深度最小的节点是唯一的[7]，**就有**$Lca(x, z) = Lca(x, y)$**或**$Lca(x, z) = Lca(y, z)$。只要先将$x_1, x_2, \ldots, x_m$按照dfs的顺序排序后，再把$Lca(x_i, x_{i+1})(1 \le i < m)$与$x_i(1 \le i \le m)$一起作为虚树中的节点即可。

有了虚树中的节点后，按照dfs序的顺序模拟这棵虚树的dfs，就能得到虚树节点之间的连边了。具体做法是，先把虚树中的节点按照原树dfs序中的顺序排序，按照这个顺序插入这些节点。维护虚树中的根到当前节点的那一条路径。当前插入点x时，依次检查这条路径上最后的那个点y是不是x的祖先[8]。**如果不是，就将**y**踢出当前的路径，继续检查；否则在**x**与**y**之间连一条边**，并将x加入路径。

上面这个构造虚树的方法时间复杂度是$O(m \log n)$的，构造出来的虚树大小是$O(m)$的。

[7]假设有两个深度相同且最小的点，那么它们的Lca也一定在这一段中，且深度更小，矛盾。

[8]可以检查x是否在整棵树中y的子树对应的那一段区间中，这是$O(1)$的。

2.2.3 操作方法

问题中的操作往往不是针对虚树中的节点的，而是针对整颗树中的节点的。在虚树中，可以将不是虚树节点的点权视为无用的。那么对一棵虚树进行某个操作，即为对那些既在这棵虚树中，又与这个操作有关的那些节点进行操作。

一般树的操作有链与子树两种，对子树的操作可以直接对dfs序维护，下面只考虑在全局中对一条链(X,Y)进行操作。这样的链一定能**拆成两条链**(X,Z)，(Y,Z)，Z是X和Y的Lca。找到X的祖先中（包括X），深度最大且为虚树中的点的节点a；找到Y的祖先中（包括Y）深度最大且为虚树中的点的节点b。若Z为虚树中的点，那么(X,Y)对应的虚树中的链为(a,b)；否则，对应的链为a到Z以下（不包括Z）与b到Z以下（不包括Z）的那两条链。可以证明，这两条链中最多只有一条是有用的，因为如果a、b的深度都大于Z，那么Z必然是虚树中的点。知道a后，可以利用倍增，找到a的祖先中，深度大于Z且深度最小的那个点c，那么(a,c)就是所要求的链。

为了找到a节点与b节点，可以事先**按深度从小到大的顺序**将虚树中的每一个节点c对应子树中的所有节点的权值改为c，这样X的权值即为a，Y的权值即为b。这些修改与询问可以利用线段树维护dfs序解决。预处理的复杂度为$O(m\log n)$，对每一个操作找到对应的虚树节点的复杂度为$O(\log n)$。

【例五】Jc的宿舍[9]

一棵n个节点的树，每一个节点有一个权值。

有Q个询问，每次询问是将一条链(x,y)上的权值存到数组w中，并从小到大排序。

若这条链上共有m个节点，那么输出$\sum_{1\le k\le m} kw_k$。

$1\le n,Q\le 50000$，强制在线。

按权值从小到大分块，建立每一个块中的虚树，并预处理出每一个块之中**两两节点之间的答案**。这只要从每一个节点出发，在虚树中dfs，并维护路径的平衡树即可。这里两两之间的答案具体指的是，**只考虑此虚树中的节点**，这两个点之间路径的答案。

[9]Author：吉如一

对于每一个询问，依次对每一个块中的虚树询问链上答案，链上节点个数以及链上权值和。链上答案在预处理时已经得到，链上节点个数与链上权值和可以预处理出节点到根路径上的和，减一下就是路径的值。

假设共有X个块，每一个块的答案，权值和分别为ans_i，w_i，前i个块的节点个数和为S_i。那么最后的答案为

$$\sum_{1\le k\le X} ans_k + w_k S_{k-1}$$

这个式子的正确性是显然的，因为某个块k中的所有节点都又有S_{k-1}个节点放在它之前。

若每个块的大小为T，那么预处理的复杂度为$O(n^2 \log n/T)$，处理每一个询问的复杂度为$O(T \log n)$。取$T = \sqrt{n}$，可以得到复杂度为$O((n+Q)\sqrt{n}\log n)$的做法。

事实上，虚树中两两之间的点数，点权和，以及每个点在虚树中对应的点，都是可以预处理的。这样每次询问复杂度变为$O(n/T)$，取$T = \sqrt{n/\log n}$，总复杂度变为$O((n+Q)\sqrt{n\log n})$。

【例六】CHANGE[10]

给定一棵大小为n的有根树，树上每一个节点有两个属性a, b，初始时所有节点的属性都相同。共有Q个操作。

操作有以下几种：

1.将一棵子树的a值修改为某个值。

2.将一棵子树的b值修改为某个值。

3.询问一条链(x, y)上，$a \in [X, Y]$的节点中，b的最大值。

由于子树修改相当于对dfs序中的某一段进行修改，因此，可以利用1.4中的方法，将其变为增加与撤销操作。但如果只这样变化的话，会给整个算法造成麻烦。考虑到每次都是修改子树，每一次操作完之后，那些转化而成的且标记相同的区间[11] **必然可以并成一个联通子图**。将这些联通子图插入到时间线段树中。

[10]题目来源：原创。

[11]这边标记的定义见1.4。

对于时间线段树中的每一个节点，建立一棵以a值为序的权值线段树，权值线段树中的每一个节点有一些修改一个联通子图b值的操作。现在考虑对权值线段树中的每一个节点怎么做。

对于每一个修改联通子图的操作，都将这个联通子图中深度最小的点，作为关键点[12]，建立包含这些关键点的虚树；并且利用线段树将这个联通子图中点的b值修改为给定值，这只要将利用1.4构造出来那些区间修改一下就可以了。

对于每一个询问，肯定能**拆成两条往祖先方向的链**。这样的有向链要么停留在某个联通子图中，要么穿过了这个联通子图中深度最小的点，到达下一个联通子图。因此，只要求出这条链通过的关键点中b值的最大值，以及这条链最后到达的那个点的b值，两者中较大的那个就是这个权值线段树中这个节点对这个询问的贡献。

简单复述一下算法：先将子树修改操作转化为联通子图增加一对(a,b)的操作，以及撤销某次增加操作的操作。将每个增加操作插入到对应的时间线段树当中。对时间线段树中的每一个节点，将包含它的增加操作取出来，按照a值建立权值线段树，并在权值线段树的每一个节点构造一棵虚树。对于这个时间线段树节点包含的询问，在权值线段树中找到$[X,Y]$对应的那些节点，并在那些节点的虚树中进行询问。

时间线段树将节点数乘上$\log Q$，权值线段树将节点数乘上$\log n$，权值线段树中的操作都是$O(\log n)$的，所以该做法的复杂度为$O(Q\log^2 n\log Q)$，这里忽略了一些在原树中预处理的复杂度。

3 总结

这种结合线段树与分治的方法对一类区间修改问题有着良好的功效。这种方法的优点有代码简单，思路清晰，高效快捷等，在实际问题中有较广的应用。结合虚树后，可以用来解决一些较难的问题。

4 特别感谢

- 感谢父母、学校对我的培养

[12]这样的点必然是唯一的。

- 感谢杜瑜皓，黄嘉泰，周子凯等同学对我的帮助
- 感谢CCF给了我一个展示自己的机会

参考文献

[1] 陈丹琦，《从〈Cash〉谈一类分治算法的应用》

[2] 许昊然，《浅谈数据结构题的几个非经典解法》

[3] 陈立杰，《JustForFun EXT 解题报告》