区间最值操作与历史最值问题

杭州学军中学 吉如一

摘 要

本文主要介绍了有关区间最值操作与历史最值问题在算法竞赛中的一系列转化与处理方法。虽然目前，有关区间最值操作、历史最值询问的问题在算法竞赛中的出现次数还比较少，但是它的出现频率正在逐渐增加，又因为选手对这一类方面涉及较少，所以在竞赛中它也有着较高的难度。本文对目前已经出现过的类似问题进行了归类分析，并介绍了一些将这一类问题转化为一些传统问题的通用方法，例如将区间最值操作转化为区间加减操作，将有关区间历史最值的和的询问转化为普通的区间和询问。希望本文可以将这一类有趣的问题带入选手们的视野中，起到抛砖引玉的作用。

1 前言

数据结构类的问题在算法竞赛竞赛中一直都是一类出现频率极高热点问题。但是因为近几年来，随着选手们对数据结构问题的研究不断深入，留给出题人的命题空间也越来越狭小。因此，在现在国内的竞赛中，大部分数据结构问题除了在不停的包装原题老生常谈外，还有一种趋势是将一些经典的序列问题放到树结构或者仙人掌结构上，强行增加选手的代码量。我认为这种行为是非常无趣的，它破坏了很多数据结构问题简洁优美的性质，让题目变得索然无味。

然而就在最近，区间最值操作（具体指区间取max操作与区间取min操作）与历史最值问题（具体指历史最大值、历史最小值与历史版本和）逐渐进入了大家的视野。实际上早在IOI2014时就出现过了有关区间最值操作的问题[1]，而历史最值问题的雏形更是能追溯到五年之前[2]，然而在那时并没有引起多少选手

[1] http://uoj.ac/problem/25
[2] http://www.tyvj.cn/p/1518

的兴趣也并没有多少人进行了深入的研究。直到去年夏天的多校联合训练，才出现了第一道难度较高的有关区间最值操作的问题[3]，之后这一类问题便渐渐多了起来，在清华集训[4]与Universal Online Judge(UOJ)[5]的比赛中都有出现。

因为这一类问题较为新颖，所以选手在初次遇到这类问题时通常难以解决。因此本文针对这一类问题，介绍了一种对区间最值操作的通用转化方法，并对常见的几类历史最值问题进行了针对性的分析，希望能对大家有所帮助。

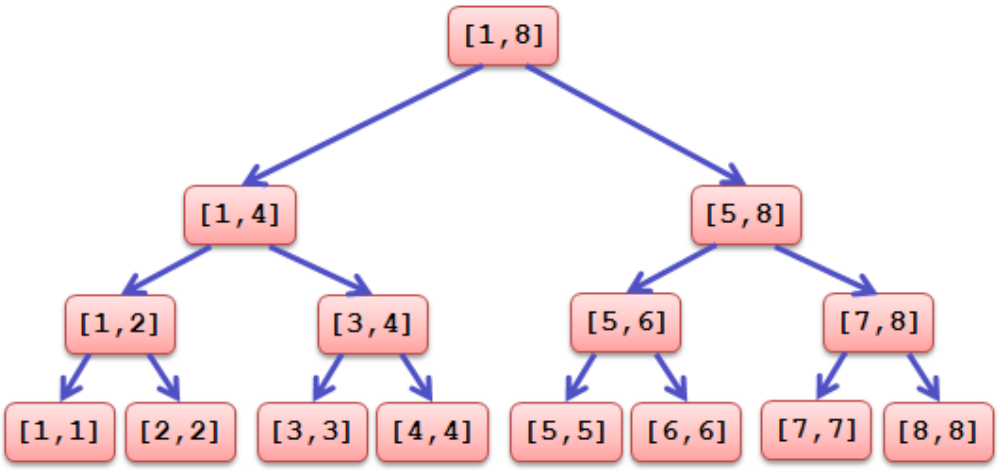
在本文的第二节中，回顾了有关线段树这一工具的定义与性质。而在第三节和第四节中，本文分别针对区间最值操作与历史最值问题进行了讨论。

2 线段树

2.1 简介

线段树是一种在算法竞赛中非常常见的数据结构。线段树的每一个节点都代表了一个区间，特别地，根节点代表总区间，叶子节点代表某个特定位置。

线段树上除了叶子节点以外的每一个节点都有两个儿子：左儿子和右儿子。如果这个节点代表的区间是$[l,r]$，那么它的左儿子代表的区间是$[l,\lfloor\frac{l+r}{2}\rfloor]$，右儿子代表的区间是$[\lfloor\frac{l+r}{2}\rfloor+1,r]$。举例来说，区间$[1,8]$构造的线段树如下图所示：



[3]http://acm.hdu.edu.cn/showproblem.php?pid=5306
[4]http://uoj.ac/problem/164
[5]http://uoj.ac/problem/169

通过利用这个结构的一些性质，我们能在线段树上快速的进行一些操作。

2.2 操作

2.2.1 单点操作

因为线段树的分治结构，孩子节点的区间长度是父节点的一半，所以线段树的树高是$O(\log n)$的。

因此对于单点操作，我们可以直接在线段树上定位这个位置所代表的叶子节点，然后更新它和它的父亲的信息，时间复杂度是单次$O(\log n)$。

2.2.2 区间询问

为了解决区间询问，我们需要在线段树上定位一个区间$[L,R]$。

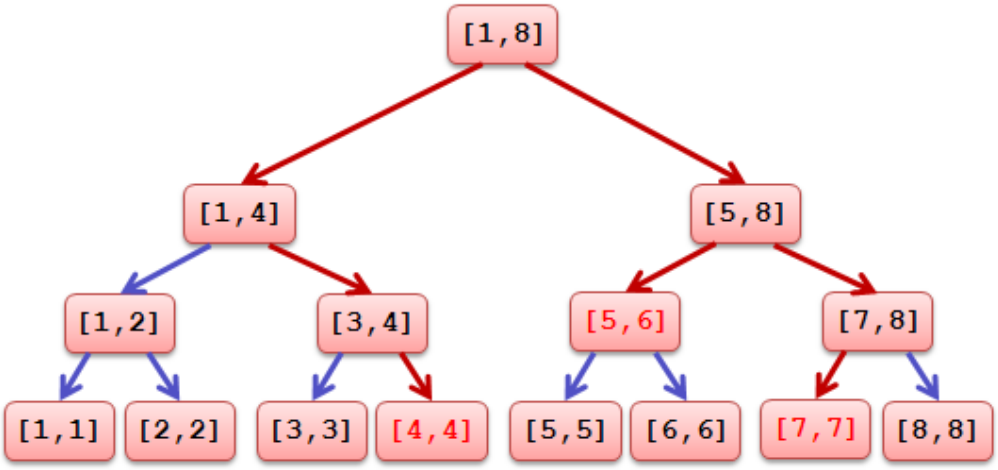
考虑这么一个算法，我们从根节点开始进行搜索，假设现在搜索到的节点的区间是$[l,r]$，那么就有三种情况：

1.如果区间$[L,R]$包含了区间$[l,r]$，即$L \le l \le r \le R$，就把它加入答案。

2.如果区间$[L,R]$与区间$[l,r]$不相交，即$L > r$或者$R < l$，那么就退出。

3.如果不满足上述两种情况，那么就分别搜索这个节点的两个孩子。

举例来说，如果我们在由区间$[1,8]$建立的线段树中定位区间$[4,7]$，那么过程如图所示（红色的边表示经过的边，红色字体的区间表示定位得到的区间）：



现在我们来证明这个做法的时间复杂度是$O(\log n)$的：

因为前两种情况是终止条件，只有在第三种情况的时候会继续递归，又因为线段树的每一个节点的儿子个数只有2，所以前两种情况的节点数不会超过第三种情况的两倍，因此我们只需要考虑第三种情况时的复杂度即可。

考虑线段树的每一层（即每一组深度相同的节点），显然这一层的所有节点所代表的区间是互不相交，我们把这些区间按照左端点排序。

考虑询问区间$[L,R]$，显然在排序后，与询问区间相交的所有区间的下标是连续的一段，在这一段中只有最左端的区间与最右端的区间才有可能满足条件三。

因为线段树的深度是$O(\log n)$的，每一层只有$O(1)$个节点满足条件三，所以这个方法的时间复杂度是$O(\log n)$的。

在线段树上定位了区间$[L,R]$后，我们只需要把这些节点的信息合并起来，就得到答案了。

2.2.3 区间修改

对于区间修改，我们自然不可能用单点修改的方法暴力修改每一个区间，这时就要引入懒标记的概念了。

以区间加减举例，我们对线段树中的每一个节点i，都记录一个懒标记A_i，表示给节点i所代表的区间中的所有数都加上了A_i。

现在有一个操作，需要给$[L,R]$加上x。我们先在线段树中定位这个区间，然后给这个区间打上懒标记x，具体来讲就是把A_i加上x并更新这个节点的区间和（加上区间大小乘上x），然后更新它的所有父节点的信息。

不难发现，对于线段树中的任意一个节点，只有当它的所有祖先节点的标记都是0的时候，它所记录的信息才是正确的。因此无论是修改操作还是询问操作，访问到每一个节点的时候，都应该将这个节点的标记下传到它的孩子中。给第i个节点进行标记下传相当于给它的左儿子与右儿子打上懒标记A_i，并把A_i清零。

时间复杂度与区间询问相同，是单次$O(\log n)$的。

懒标记的用处非常大，只要标记可以合并、在打上标记的时候可以快速更新线段树节点的信息，那么就能使用线段树维护，例如区间加减询问区间和、区间覆盖询问区间和、区间覆盖询问区间gcd等等。但是同样它也存在局限性，

例如区间对一个数取max询问区间和，使用基本的懒标记方法就是无法处理的。

3 区间最值操作

对一个序列A，区间最值操作具体指，给出三个数l, r, x，对所有的$i \in [l, r]$，把A_i变成$\max(A_i, x)$或者$\min(A_i, x)$。

接下来我们通过几道例题来对这类问题的处理方法进行讨论。

例题1. *Gorgeous Sequence*[6]

给出一个长度为n的数组A，接下来有m次操作，操作有如下三种：

*1.*给出l, r, x，对所有的$i \in [l, r]$，把A_i变成$\min(A_i, x)$。

*2.*给出l, r，对所有的$i \in [l, r]$，询问A_i的最大值。

*3.*给出l, r，询问$\sum_{i=l}^{r} A_i$。

数据范围：$n, m \leq 10^6$

就如第一节中所说明的，因为在给一个节点打上区间取min标记的时候我们无法快速更新区间和，所以这一个问题是无法通过传统的懒标记来解决的。

考虑下面这一种解法：对线段树中的每一个节点除了维护区间和num以外，还要额外维护区间中的最大值ma、严格次大值se以及最大值个数t。

现在假设我们要让区间$[L, R]$对x取min，我们先在线段树中定位这个区间，对定位的每一个节点，我们开始暴力搜索。搜索到每一个节点时候我们分三种情况讨论：

1.当$ma \leq x$时，显然这一次修改不会对这个节点产生影响，直接退出。

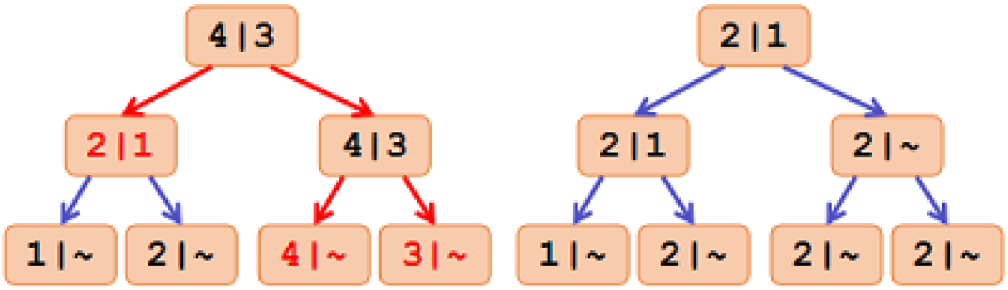
2.当$se < x < ma$时，显然这一次修改只会影响到所有最大值，所以我们把num加上$t \cdot (x - ma)$，把ma更新为x，接着打上标记然后退出。

3.当$se \geq x$时，我们无法直接更新这一个节点的信息，因此在这时，我们对当前节点的左儿子与右儿子进行递归搜索。

如下图所示，左图是一棵建立在区间$[1, 4]$上的线段树，每一个节点记录的信息的左侧是区间最大值，右侧是严格次大值。现在我们要让区间$[1, 4]$对2取min。

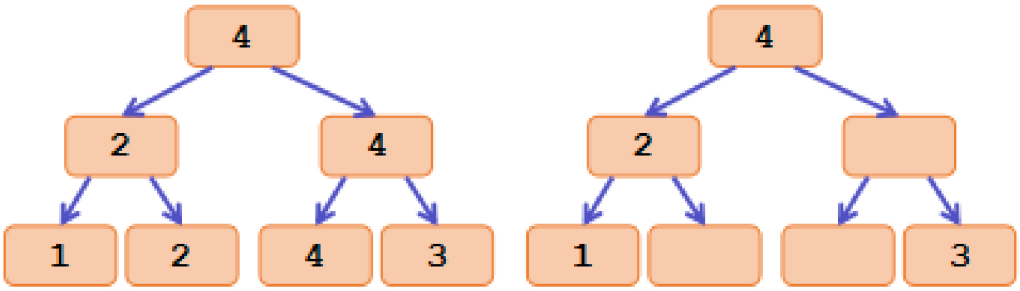
[6]Author：徐寅展，题目可以在这里找到：http://acm.hdu.edu.cn/showproblem.php?pid=5306

那么左图中的红色边表示在搜索时经过的边，红色字体的节点表示搜索终止的节点，右图为更新后的线段树。



如果进行实现的话，我们可以发现这一个算法的运行效率非常高。实际上我们可以证明它的时间复杂度是$O(m \log n)$的。

首先我们把最大值看成标记，对线段树每一个节点，记录一个标记值，它的值等于它所控制的区间中的最大值。接着如果一个点的标记值和它父亲的标记值相同，那么我们就把它这个位置的标记删去，大致转化如下图所示（左图记录的是线段树中的最大值，右图是转化后每一个位置的标记）：



显然在转化之后，线段树中最多存在n个标记。这些标记满足每一个标记的值都大于它子树中的所有标记的值。每一个位置实际的值等于从它对应的线段树的叶子节点出发，向上走遇到的第一个标记的值。

观察上述算法，可以发现我们维护的区间次大值，相当于子树中标记的最大值。而暴力DFS的作用，相当于在打上了新的标记后，为了满足标记值随深

度递减的性质，在子树中回收掉值大于它的标记，即标记回收。现在我们要证明的是标记回收的复杂度。

首先我们定义标记类的概念：

1.同一次区间取min标记产生的标记属于同一类。

2.同一个标记下传产生的新标记属于同一类。

3.不满足前两个条件的任意两个标记属于不同类。

接着定义每一个标记类的权值等于这一类标记加上线段树的根节点在线段树上形成的虚树大小（即所有子树中存在这一类标记的点的个数），定义势函数$\Phi(x)$为线段树中的所有标记类的权值总和。

考虑一次区间取max操作，我们添加了一个新的标记类，它的权值等于我们打标记时经过的点数，这显然单次是$O(\log n)$。

考虑一次标记下传，显然我们只让这一类标记的权值增加了$O(1)$。

考虑DFS的过程，我们一旦开始进行暴力DFS，那么说明这个点的子树中一定存在需要回收的标记，而在回收之后这个点的子树中一定不存在这一类标记，所以我们经过每一个节点，一定至少存在一类标记的权值减少了1，那么必定伴随着势函数减少了1。

因为标记下传只发生于我们在线段树上定位区间的时候，所以标记下传的总次数是$O(m\log n)$的，而打标记时对势函数产生的总贡献也是$O(m\log n)$的，所以势函数总的变化量只有$O(m\log n)$。

到此，我们就证明了这个算法的复杂度是$O(m\log n)$的。

例题2. *Picks loves segment tree*[7]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i\in[l,r]$，将A_i变成$\min(A_i,x)$。

2. 对于所有的$i\in[l,r]$，将A_i加上x（x可能为负数）。

3. 对于所有的$i\in[l,r]$，询问A_i的和。

数据范围$n,m\leq 3\times 10^5$。

因为加上了区间加减，区间最大值、严格次大值和最大值个数还是可以维护的，所以我们考虑沿用刚才的做法。

接着我们来证明复杂度。因为有了区间加减这一个操作，例题一的证明就难以沿用了，但是大致的思想可以借鉴。

[7]题目来源：原创

我们可以修改一下标记类的定义：

1.同一次区间取min标记产生的标记属于同一类。

2.同一个标记下传产生的新标记都属于原来的一类。

3.对于一次区间加减操作，对于当前线段树中的任意一类，如果它被修改区间完全包含或者和修改区间完全相离，那么这一类就不变；否则我们就让这一类分裂成两类：一类是被修改的部分，一类是剩下的部分。

重新定义每一类标记的权值：等于这一类标记在线段树上所有节点形成的虚树大小（在这儿不包含根节点）。定义辅助函数$\Phi(x)$等于所有标记类的权值和。

这样一次区间加减操作对辅助函数函数的影响只有单次$O(\log n)$的标记下传（分裂标记的时候势能函数只会减少）；一次标记下传只会让辅助函数增加$O(1)$；一次打标记只会让辅助函数增加$O(\log n)$。

考虑DFS的过程，我们先通过线段树定位了若干个节点，然后从这些节点开始DFS。我们对每一个开始节点考虑每一类会被回收的标记：如果这一类标记只有一部分在这个子树中，那么我们每经过一个节点都会让这一种标记的权值减一；而如果这一类标记全部都在子树中，我们回收的过程相当于先花了若干的代价定位了这一类标记的LCA，然后再在标记的虚树上进行回收。

所以我们可以把复杂度分成两部分，第一部分是减少辅助函数时花费的时间复杂度，这是$O(m\log n)$的；第二部分的复杂度是定位某一类标记的LCA时产生的复杂度，每一类标记都可能对这一部分的复杂度产生$O(\log n)$的贡献，因为标记的总数只有$O(m\log n)$，所以标记的类数也是$O(m\log n)$的，因此这一部分的复杂度的一个上界是$O(m\log^2 n)$的。

到此我们就证明了这一个算法解决这个问题时的复杂度是$O(m\log^2 n)$的（当然，如果使用标记深度和作为势函数可以更方便的得到这个上界，这儿主要是沿用例题一的证明）。

实际上这个上界是比较松的，对目前我构造的数据来说，这个算法的运行效率都更接近与$O(m\log n)$。因此我猜想在这题的证明中标记的总类数是线性的，即这一个算法的时间复杂度是$O(m\log n)$的，然而到目前为止还没有好的方法能够证明，也无法构造出卡到$O(m\log^2 n)$的数据。所以在接下来的讨论中，我们就使用$O(m\log^2 n)$来表示这个算法的复杂度。

3.1 小结

通过上述两道例题，我们得到了一种在区间最值操作中维护区间和的方法（区间取max操作与区间取min操作类似）。

观察例题二的复杂度证明，我们发现在证明的时候并没有使用到区间加减操作的具体性质。因此我们将区间加减操作给替换成其他的修改操作，这一证明还是成立的。

例题3. *AcrossTheSky loves segment tree*[8]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\min(A_i, x)$。
2. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i, x)$。
3. 对于所有的$i \in [l, r]$，询问A_i的和。

数据范围$n, m \leq 3 \times 10^5$。

在我们单独解决区间取min操作与区间取max操作的时候，我们需要维护区间最小值、区间严格次小值、区间最小值个数、区间最大值、区间严格次大值以及区间最大值个数，现在这两个操作同时出现，但是这些信息还是可以维护的。

分析复杂度的时候可以把两种标记分开来考虑，显然每种标记对另一种标记的影响只有标记下传（在极端情况的时候还有可能直接回收掉某一棵子树里的部分标记，但是因为这个回收的时间复杂度是$O(1)$的，所以不会产生影响），因此例题二中的证明依然还是成立的，于是这个算法的时间复杂度应当是$O(m \log^2 n)$。

3.2 将区间最值操作转化为区间加减操作

观察我们解决例题二时使用的算法，我们发现在打上区间取min标记时，实际上这一个区间中只有最大值会受到这一个标记的影响，而其他数都是不会变的。同时在最大值被修改之后，原来是区间最大值的所有数依然还是区间最大值，原来不是区间最大值的所有数依然不是区间最大值。

所以实际上，通过上述算法，我们把区间最值操作，转化成了一种只针对区间最大（小）值进行加减修改的操作。

[8]题目来源：原创

因此我们可以把线段树中的每一个节点所维护的区间拆成两类，一类是这个区间中的最大值，一类是这个区间中的其他数。具体操作可以用例题二举例，在例题二中，我们需要处理的操作有：对一个线段树节点内的所有最大值加上或者减去一个数，对一个区间内的所有数加上或者减去一个数。因此我们只需要对线段树中的每一个节点中的最大值与非最大值分别维护标记与信息即可。

在标记下传的时候，需要根据两个儿子中的最大值情况进行讨论，最大值的标记只能下传到最大值较大的那个孩子中（最大值相同的时候同时下传）。

到此，我们就得到了一种在支付一个常数较小的$\log n$的复杂度的情况下，将区间最值操作转化为区间加减操作的方法。接下来我们通过三道不同方面的例题来进一步了解这个方法：

例题4. *Mzl loves segment tree*[9]

给出一个长度为n的数列A，定义一个数组B，最开始数组B的所有数都是0。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\min(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i, x)$。

3. 对于所有的$i \in [l, r]$，将A_i加上x。

4. 对于所有的$i \in [l, r]$，询问B_i的和。

在每次操作之后，对于每一个位置i，如果A_i的值发生了变化，那么就给B_i加上一，否则B_i不变。

数据范围$n, m \leq 3 \times 10^5$。

如果只有区间加减操作，那么只要$x \neq 0$，那么每次修改的区间内的所有数都会发生变化，只要给B_i区间加上一就好了。

现在有了区间取min操作和区间取max操作，我们可以把线段树中的每一个节点中的数分成三个部分：最大值、最小值、既不是最大值也不是最小值的数，接着对每一个类数分别维护，这样就能消除区间取min操作与区间取max操作了。

值得注意的是当两个最值操作同时存在的时候，最大值与最小值所控制的数集可能会发生重叠，在这种情况下需要特殊处理避免在答案中重复统计某一个B_i的值。

[9]Author：黄志翱，这儿略有改编。

例题5. *ChiTuShaoNian loves segment tree*[10]

给出两个长度为n的数列A和B。接下来进行了m次操作，操作有五种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\min(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将B_i变成$\min(A_i, x)$。

3. 对于所有的$i \in [l, r]$，将A_i加上x。

4. 对于所有的$i \in [l, r]$，将B_i加上x。

5. 对于所有的$i \in [l, r]$，询问$A_i + B_i$的最大值。

数据范围：$n, m \leq 3 \times 10^5$

在这个问题中有两个操作的对象：数组A与数组B。在只有单一数组的时候，我们需要在每一个节点中把位置分成两类：区间最大值与非区间最大值。

稍微拓展一下一下，便可以得到这题的做法了。我们在线段树上，把每一个节点中的数给分成四类：同时在A, B中是区间最大值的位置，在A中是区间最大值而在B中不是的位置，在B中是区间最大值而在A中不是的位置，同时不是A, B中的最大值的位置。然后对这四类数分别记录答案与标记即可。

至于复杂度，我们发现两个数组在操作中是完全独立的，两套标记之间并不会产生任何影响，因此时间复杂度还是$O(m \log^2 n)$。

不难发现这个做法拓展到多个数组也是可行的：当我们对K个数组同时操作的时候，我们需要把每一个节点中的数分成2^K类，因此时间复杂度是$O(2^K \cdot m \log^2 n)$，空间复杂度是$O(n2^K)$的。

例题6. *Dzy loves segment tree*[11]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\min(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将A_i加上x。

3. 询问$\gcd(A_l, A_{l+1}, ..., A_{r-1}, A_r)$。

数据范围：$n, m \leq 10^5$

考虑只有区间加减操作的情况，这是一个经典问题，因为$\gcd(a, b, c) = \gcd(a, b - a, c - b)$，所以我们可以给数组$A$进行差分（即用$A_i - A_{i-1}$来替代$A_i$）。

[10]题目来源：原创

[11]题目来源：原创

在差分之后问题就转化成了两部分：第一部分是区间加减询问单点值，第二部分是单点修改询问区间gcd。这两部分都可以用线段树简单的维护。

现在我们来考虑有了区间取min操作后的情况，因为我们对区间最值操作的处理方法是把数拆成两类，但是这样的话数的顺序就会被打乱了，不能直接按照之前的下标顺序进行差分。于是我们考虑用另一种差分方式：把区间最大值单独拿出来维护，将非最大值的所有数进行差分。

具体来说，我们对线段树中的每一个节点额外维护以下的东西：最大值ma、严格次大值se，非最大值序列的开头s、结尾t、差分之后这个序列的最大公约数num。

现在考虑从两个儿子l与r中更新信息，这儿我们只讨论一种情况：$ma_l > ma_r$。在这时显然有$ma = ma_l, se = \max(se_l, ma_r)$。考虑更新差分序列的信息，注意到差分顺序可以是任意的，我们按照左子树序列、右子树序列、右子树最大值的顺序把非最大值序列给拼接起来，即让$s = s_l, t = ma_r, num = \gcd(num_l, num_r, t_l - s_r, t_r - ma_r)$。其他两种情况与这种类似，就不再赘述了。

考虑打上一个区间加减标记，因为我们将非最大值组成的序列差分了，所以num是不会被影响的，只要修改一下其他信息就行了，这是非常简单的。

如果算上gcd的复杂度的话，我们就得到了一个$O(m \log^3 n)$的做法。

3.3 总结

在这一节中，我们得到了一个处理区间最值操作的通用方法：通过维护最大（小）值与严格次大（小）值，在支付一个常数较小的$\log n$的时间复杂度内，将区间最值操作转化为区间加减操作。

但是这样的做法也有一定的局限性。从例题六中可以看出，当涉及到的询问与下标顺序有关的时候，这个做法就难以适用了，因为在这个做法中，我们把一个节点拆开的时候是没有考虑下标的顺序的。例如区间加减，区间取max，询问某一个区间的所有子区间中，权值和的最大值，这一个问题用上述的方法就难以解决。

4 历史最值问题

4.1 简介

在数据结构问题中，我们通常需要对一个给定的数组A进行若干次操作，然后进行一些询问。

目前大多数数据结构题都是对当前版本进行询问，也有一少部分数据结构题需要对历史版本进行询问，这一部分题目中的绝大部分考察的都是可持久化数据结构方面的知识，除此之外，有一类特殊的有关历史版本的问题我们把它称作历史最值问题。

历史最值问题中的询问大致可以分为以下三类：

4.1.1 历史最大值

定义一个辅助数组B，最开始B数组与A数组完全相同。在每一次操作后，对每一个$i \in [1,n]$，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。这时，我们将B_i称作i这个位置的历史最大值。

4.1.2 历史最小值

定义一个辅助数组B，最开始B数组与A数组完全相同。在每一次操作后，对每一个$i \in [1,n]$，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。这时，我们将B_i称作i这个位置的历史最小值。

4.1.3 历史版本和

定义一个辅助数组B，最开始B数组中的所有数都是0。在每一次操作后，对每一个$i \in [1,n]$，我们都进行一次更新，让B_i加上A_i。这时，我们将B_i称作i这个位置的历史版本和。

接下来，我们将历史最值问题按照难度从低到高分成四类进行讨论。

4.2 可以用懒标记处理的问题

例题7. *CPU*监控[12]

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成x。

2. 对于所有的$i \in [l, r]$，将A_i加上x。

3. 对于所有的$i \in [l, r]$，询问A_i的最大值。

4. 对于所有的$i \in [l, r]$，询问B_i的最大值。

在每一次操作后，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。

数据范围：$n, m \leq 10^5$

刚接触这一类问题时，这个例题的难度可能较高，所以我们先忽略第一种操作。

考虑使用传统的懒标记来解决，首先如果只是询问区间最大值，只需要使用区间加减这一个懒标记（用Add表示）就能解决。

现在考虑询问区间历史最大值的最大值。我们定义一种新的懒标记：历史最大的加减标记（用Pre表示）。这个标记的定义是：从上一次把这个节点的标记下传的时刻到当前时刻这一时间段中，这个节点中的Add标记值到达过的最大值。

现在考虑把第i个节点的标记下传到它的儿子l，不难发现标记是可以合并的：$\mathrm{Pre}_l = \max(\mathrm{Pre}_l, \mathrm{Add}_l + \mathrm{Pre}_i), \mathrm{Add}_l = \mathrm{Add}_l + \mathrm{Add}_i$。至于区间历史最大值信息的更新也与标记的合并类似，只需要将当前的区间最大值加上Pre_i然后与原来的历史最大值进行比较即可。

现在回到原题，我们观察在修改操作过程中，被影响到的节点的变化：如果一个节点没有发生标记下传，那么最开始它一直被区间加减操作所影响，这时我们可以用上面描述的Pre标记来记录，直到某一时刻，这个节点被区间覆盖标记影响了，那么这时这个节点中的所有数都变得完全相同，再之后的所有区间加减修改，对这个节点来说，与区间覆盖操作并没有不同。

因此每一个节点受到的标记可以分成两个部分：第一个部分是区间加减，第二个部分是区间覆盖。因此我们可以用(x, y)来表示历史最值标记，它的定义

[12]题目来源：http://www.tyvj.cn/p/1518

是当前区间在第一阶段时最大的加减标记是x，在第二个阶段时最大的覆盖标记是y。显然这个标记是可以进行合并与更新的。

到此我们就使用最传统的懒标记方法解决了这个问题，时间复杂度$O(m \log n)$。

例题8. V^{13}

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有五种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i - x, 0)$。
2. 对于所有的$i \in [l, r]$，将A_i加上x。
3. 对于所有的$i \in [l, r]$，将变成x。
4. 给出一个i，询问A_i的值。
5. 给出一个i，询问B_i的值。

在每一次操作后，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。

数据范围：$n, m \leq 5 \times 10^5$

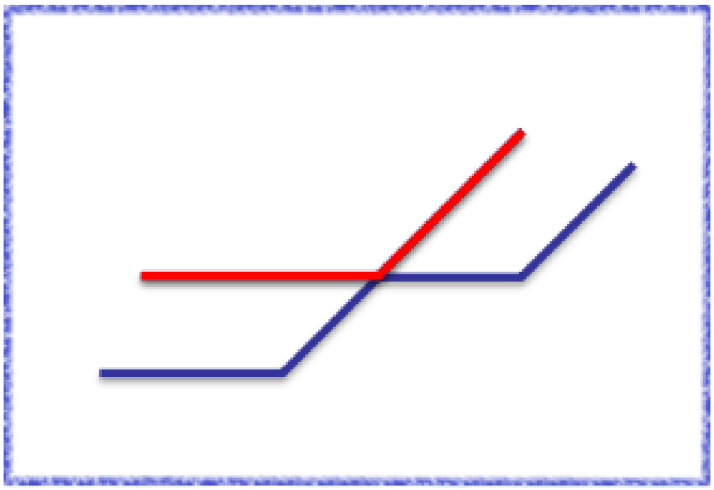
首先我们定义一种标记(a, b)，表示给这个区间中的所有数先加上a，然后再对b取max。不难发现题目中涉及的三种操作都可以用这一个标记来表示，分别为$(-x, 0), (x, -\text{inf}), (-\text{inf}, x)$，其中inf是一个事先设定的足够大数。

考虑合并两个标记(a, b)与(c, d)，那么显然得到的就是$(a + c, \max(b + c, d))$。因此对于询问4，我们只需要把询问的叶子节点到根路径上的所有标记都下传到叶子节点，就能直接求得答案了。

现在考虑历史最大标记。我们可以把标记给看成一个函数，每一个数字对应的函数值表示这个数在作用了这个标记后的答案，在这个问题中，标记(a, b)就相当于是一个分段函数，第一段是一条与x轴平行的直线，第二段是一条斜率为1的直线。

在更新历史最大标记的时候，我们相当于把两个标记的图象同时放到一个坐标系中，然后取出上方的那一段。下图展示的是标记合并时可能出现的一种情况，蓝色的部分是原来的两个标记，红色部分是更新后得到的标记。

[13]Author：彭雨翔，题目可以在这里找到：`http://uoj.ac/problem/164`



因为在更新标记之后形式和原来相同，所以这个历史最值标记是可以维护的。因此对于询问5，我们也只需要把所有相关的标记给下传下去，就能得到答案了。

时间复杂度$O(m \log n)$。

例题9. *Rikka with Sequences*[14]

给出一个长度为n的数列A，同时定义一个$n \times n$的辅助数组B，最开始$B_{i,j} = \sum_{k=i}^{j} A_k$。接下来进行了$m$次操作，操作有两种类型：

1. 给出两个整数l和x，把A_l的值变成x。
2. 给出两个整数l和r，保证$l < r$，表示询问$B_{l,r}$的值。

在每次操作之后，我们都会进行一次更新：$B_{i,j} = \min(B_{i,j}, \sum_{k=i}^{j} A_k)$。

数据范围：$n, m \leq 10^5$

因为这题不方便使用在线的方式处理，所以我们可以来考虑离线的算法。

首先我们把所有询问都读进来，并把每一个询问(l, r)都看成二维平面上的一个点。考虑一次对位置i的修改，它会对所有横坐标小于等于i，纵坐标大于等于i的询问产生影响。

所以现在问题就变成了，平面上有一些点，接下来有一些操作，每一次会给一个矩形区域内的所有点加上一个数，或者询问一个点权值的历史最小值。

于是这题相当于是二维情况下的历史最值问题，一维的情况我们可以轻松的用线段树来解决，因此一个非常自然的想法就是使用二维线段树。但是由于

[14]题目来源：原创

历史最小值的标记是时间依赖的，并不能拆成若干个部分，所以二维线段树并不适用。

为了解决这个问题，我们需要一个满足以下条件的数据结构：

1.标记的作用必须是以时间为顺序的。

2.影响到同一个点的标记必须影响在数据结构的同一个点上。

在二维情况下满足以上条件的一个数据结构就是kdtree了。所以我们可以先对原来的所有点建出kdtree，然后打标记的时候和线段树类似：对kdtree 每一个点都维护一个矩形区域，然后如果和标记区域相离就退出，被包含就直接打标记，否则就递归。在询问的时候和线段树一样，把标记下传下去即可。

时间复杂度$O(m\sqrt{n})$

4.3 无法用懒标记处理的单点问题

这儿的单点问题除了询问单点的历史最值外，也包括区间询问历史最小值的最小值与区间询问历史最大值的最大值，因为这两个区间询问的处理方法与单点询问完全相同，所以把它们归入单点问题之中。

例题10. 元旦老人与数列[15]

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成$\max(A_i,x)$。

2. 对于所有的$i \in [l,r]$，将A_i加上x。

3. 对于所有的$i \in [l,r]$，询问A_i的最小值。

4. 对于所有的$i \in [l,r]$，询问B_i的最小值。

在每一次操作后，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。

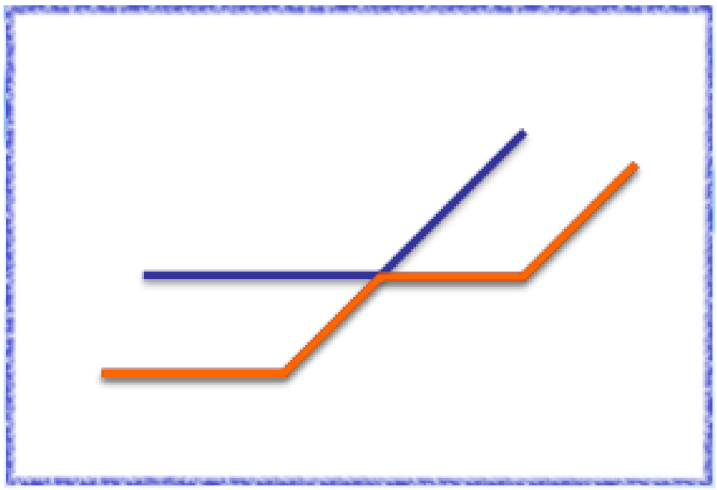
数据范围：$n,m \leq 3 \times 10^5$。

这儿涉及到了区间最值操作，区间最值操作在第三节中我们已经进行了深入的研究。

因为在这个问题中，区间最值操作的方向与历史最值询问的方向是相反的（一个是取max，一个是历史最小值），所以标记无法合并（如下图所示，两个

[15]题目来源：原创，题目可以在这里找到：http://uoj.ac/problem/169

蓝色标记合并出来的部分是下方的橙色曲线，它的段数会随着更新次数线性增长，这是不能接受的），因此我们无法沿用例题8的懒标记做法。



在第三节中我们已经得到了一种把区间最值操作转化为区间加减操作的方法，在这儿我们可以沿用。我们把线段树中每一个节点所控制的位置分成最小值与非最小值两部分，这样要处理的就只有区间加减操作了：这时的历史最值标记是可以轻易维护的。

到此我们就解决了这个问题，时间复杂度$O(m\log^2 n)$。

4.4 无区间最值操作的区间问题

这一标题所指的问题主要是以下三个：区间加减，询问区间历史最小值的和；区间加减，询问区间历史最大值的和；区间加减，询问区间历史版本和的和。

这三个问题都不能使用懒标记来进行维护，而且也不像上一节的区间最值操作一样被我们所熟知。接下来本文将分别对这三个问题介绍转化方法。

4.4.1 历史最小值

设A_i为当前数组，B_i为历史最小值。我们定义一个辅助数组C_i，每一时刻都让$C_i = A_i - B_i$。

那么我们对A_i加上数x时，如果B_i没有发生变化，那么相当于给C_i加上了x，否则C_i变成了0。具体来说，区间加减操作相当于将C_i变成了$\max(C_i+x,0)$——这是一个我们非常熟悉的操作。

考虑区间和这一询问，只要能够求出A_i和C_i的区间和，直接相减便能得到答案。前者是非常基础的线段树问题，后者我们早已在第三节中研究过了。到此我们就得到了一种$O(m\log^2 n)$的算法。

4.4.2 历史最大值

处理方法与历史最小值类似。设A_i为当前数组，B_i为历史最小值。我们同样定义一个辅助数组C_i，每一时刻都让$C_i=A_i-B_i$。这时区间加减操作相当于将C_i变成$\min(C_i+x,0)$，直接求和即可。

时间复杂度$O(m\log^2 n)$。

4.4.3 历史版本和

设A_i为当前数组，B_i为历史版本和，同时我们让t为当前结束的操作数（不包括当期操作）。同样，我们考虑定义个辅助数组C_i，每一时刻都让$C_i=B_i-t\cdot A_i$。

不难发现，对于区间加减没有影响到的位置，B_i都不会发生变化。当我们给A_i加上x的时候，我们相当于给C_i减去了$x\cdot t$。于是我们就把历史版本和转化成了简单的区间加减询问区间和的问题，直接懒标记就好了。

时间复杂度$O(m\log n)$。

4.5 有区间最值操作的区间问题

经过上面这些讨论后，这最后一类的算法已经有了雏形：只需要先通过第三节中的方法把区间最值操作转化成区间加减操作，再使用上一小节中的方法来进行求解就能够得到答案了。

因为这一类问题的处理方法比较类似，我们这儿就只选取其中一种作为例题进行分析。

例题11. 赛格蒙特彼茨[16]

[16]题目来源：原创

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将A_i加上x。

3. 对于所有的$i \in [l, r]$，询问B_i的和。

在每一次操作后，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。

数据范围：$n, m \leq 10^5$。

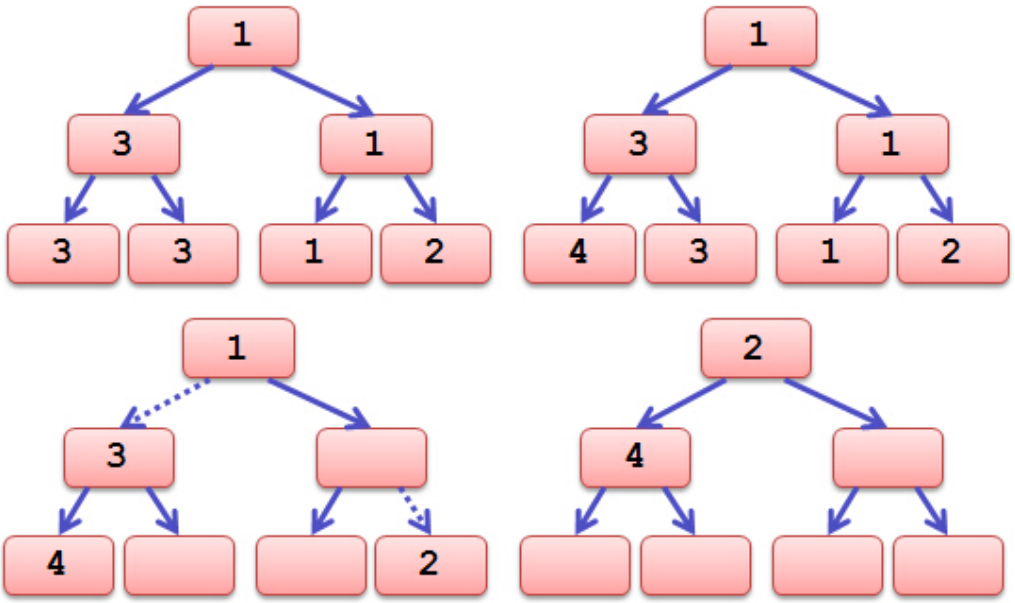
首先，我们使用上一小节中的方法，把询问转化成维护A数组和C数组的区间和。接着我们用第三节中的方法，对数组A维护区间最小值和次小值，从而把区间取max 操作转化成对区间最小值的加减操作，这样就能维护数组A的区间和了。

然后，我们来考虑怎么对C维护区间和，影响到C的操作有：对所有$i \in [l, r]$，把C_i变成$\max(C_i + x, 0)$（接下来称为第一类操作）；对所有$i \in [l, r]$，如果A_i是这个区间中A的最小值，那么把C_i变成$\max(C_i + x, 0)$（接下来称为第二类操作）。

所以还是使用第三节的方法，我们对线段树的每一个节点维护所有A的值是最小值的位置中C_i的最小值、次小值、最小值个数以及所有A的值不是最小值的位置中C_i的最小值、次小值、最小值个数，然后在对C进行修改的时候分别进行暴力DFS就行了。

最后我们来证明这个算法的时间复杂度。我们把A数组的部分称为第一层，C数组的称为第二层。首先第一层的复杂度和第三节的时候一样，我们能够证明出它是$O(m \log^2 n)$的，所以我们接下来只考虑第二部分的复杂度。

仿照第三节中的方法，首先我们先把第二层的最小值给转化成标记。可以对A的值等于最小值的所有位置以及A的值不等于最小值的所有位置分别转化成标记（接下来我们把这两个分别称为第一棵树和第二棵树），转化之后的树如下图所示（上方是原树，两侧分别为A和C的最小值，下方左图是第一棵树，右图是第二棵树）：



可以发现任何时刻，任何一个出现在第二棵树中的标记一定在第一棵树中出现过，所以两棵树上出现过的标记总数的数量级和第一棵树是相同的，所以我们只需要对第一棵树进行分析就行了。

因为第一棵树的定义是依赖数组A的，所以如果在原树中如果某一个节点的最小值和它的父节点不同，那么在第一棵树中这个位置到它父亲的转移是不存在的，我们需要删掉这条边（在图中用虚线表示）。因此在转化后第一棵树被剖成了若干个联通块，每一个联通块都至少存在一个叶子节点且满足标记值随着深度的增加而递增的性质。

因为直接对第一棵树进行标记回收以及区间加减的时候和A数组并没有什么区别，所以我们只考虑A数组中进行暴力DFS的时候产生的影响。在暴力DFS的时候，我们不仅对第一棵树进行了若干次子树加减操作，而且还合并了若干个联通块。不过如果我们在DFS的同时对标记进行下传，那么在合并两个联通块的时候，处在下方的联通块的根节点和上方联通块的根节点之间的路径上应该没有任何标记，所以这时直接把两个联通块合并起来，是不影响标记随着深度递增的限制的。

在第三节中，我们证明了暴力DFS的时候经过的节点的数的一个上界是$O(m\log^2 n)$，因此在第一棵树中标记下传的次数的上界是$O(m\log^2 n)$的，因

此标记总数是$O(m\log^2 n)$的。沿用算法三中的证明，我们可以得到这个算法的一个复杂度上界$O(m\log^3 n)$。

不过不难发现这个上界是非常松的，换句话说常数非常小，实际上在目前构造出来的数据中，暴力DFS的次数都远远没有达到$O(m\log^3 n)$的级别，因此我猜想应该存在更低的上界的证明（实际上如果第三节中我的猜想得到了证明的话，沿用上述证明就可以得到一个$O(m\log^2 n)$的上界），但是因为目前我还没有得到更好的证明方法或者极限数据构造方法，所以目前只能用$O(m\log^3 n)$ 这样一个比较松的上界来衡量这个算法的运行效率。

到此，我们就得到了一个$O(m\log^3 n)$的算法来解决这题。实际上这题就是第三节中的算法的嵌套，不难发现，上述的证明对更多层的嵌套依然有用，即如果我们有K个数组进行嵌套，那么沿用上述算法可以得到一个$O(2^K\cdot m\log^{K+1} n)$复杂度的算法。

4.6 总结

在这一节中，我们针对三种历史最值问题进行了深入的讨论，并介绍了一种通过构造辅助数组把历史最值的区间和这一询问转化为简单的区间和询问的方法。

在目前国内的OI竞赛中，出现过的历史最值相关的问题非常有限，而且大部分都集中在难度较低的前两类中。因此有关这一类问题的挖掘应当还有很大的空间，我所解决的只是其中的一小部分，例如区间加减，询问区间和的历史最小值这一问题，到目前为止我也没有好的解决方案，因此，我希望这篇文章能起到抛砖引玉的作用。同时，如果有人对这一类问题有了新的收获，也欢迎来与我交流。

5 致谢

1. 感谢中国计算机学会提供学习和交流的平台。

2. 感谢学军中学中学的徐先友老师的关心和指导。

3. 感谢徐寅展学长给予的启发与指导。

4. 感谢彭雨翔学长、吕凯风学长、毛啸同学、罗哲正同学给予的帮助。

5. 感谢周子鑫同学、毛潇涵同学的验稿。

6. 感谢其他所有本文所参考过的资料的提供者。

7. 感谢各位百忙之中抽出宝贵的时间阅读。

参考文献

[1] 徐寅展，IOI2014国家集训队论文《线段树在一类分治问题上的应用》

[2] 线段树的英文维基百科

https://en.wikipedia.org/wiki/Segment_tree

[3] 多校联合训练 Gorgeous Sequence 题解

http://blog.sina.com.cn/s/blog_15139f1a10102vnzl.html