从Unknown谈一类支持末尾插入删除的区间信息维护方法

安徽师范大学附属中学 罗哲正

摘 要

Unknown是我的集训队互测题“我们仍未知道那天所看见的数据结构的名字”。这一类要求支持末尾插入删除以及区间查询的问题，近年来在信息学竞赛中时常出现。本文将通过详细介绍Unknown一题一步步的求解过程，对这一类问题几个不同的分支进行归类分析，总结并提出一些通用算法，并对算法的性能进行分析。本文的核心是总结提出了在不容易快速合并信息的情况下，处理在端点处插入删除并维护区间信息的通用方法。希望对大家有所帮助，并能引发大家对这一类问题的思考。

1 试题大意[1]

有一个元素为向量的序列S，下标从1开始，初始时S为空，现在你需要支持三个操作：

1.在S的末尾添加一个元素(x,y)。

2.删除S的末尾元素。

3.询问下标在$[l,r]$区间内的元素中，$(x,y)\times S_i$的最大值。

其中×表示向量的叉积，$(x_1,y_1)\times(x_2,y_2)=x_1y_2-x_2y_1$

2 输入格式

第一行一个整数tp表示数据类型，以下多组数据（不超过3组），以$m=0$结束。

[1]本题改编自SDOI2014向量集 http://www.lydsy.com/JudgeOnline/problem.php?id=3533

对于每组数据，第一行一个整数m表示操作数。

接下来m行，每行有三种形式：

1 x y 在S末尾添加元素(x,y)。

2 删除S末尾元素。

3 l r x y 询问下标在$[l,r]$区间内的元素中，$(x,y)\times S_i$的最大值。

3 输出格式

对于每组数据，你需要把每个询问的结果对$M=998244353(7*17*2^{23}+1$,一个质数)取模之后异或起来输出。

注意数学意义上的取模结果在$[0,M)$区间内，例如$-1 \mod M = M-1$。

4 数据范围与约定

各测试点数据范围见下表：

测试点编号	m的规模	其他
1	$m \le 1000$	无
2		
3	$m \le 80000$	
4	$m \le 300000$	无2操作，3操作的询问为全部区间
5		所有2操作在1操作的后面，3操作的询问为全部区间
6		所有3操作都在1操作和2操作后面
7	$m \le 500000$	对于所有3操作有$l=1$，内存限制为128M
8		无
9		内存限制为128M
10		内存限制为64M

对于全部数据，令n为任意时刻序列长度，$n \le 300000; m \le 500000$。

1操作个数不超过300000，且输入的x,y满足$-10^9 \le x \le 10^9; 1 \le y \le 10^9$。

3操作个数不超过300000，且输入的x,y满足$1 \le x \le 10^9; -10^9 \le y \le 10^9$。

3操作中，l,r满足$1 \le l \le r \le n$。

时间限制为5s(Tsinsen)/3s(UOJ)，内存限制为512MB.

5 算法介绍

5.1 算法1：暴力枚举

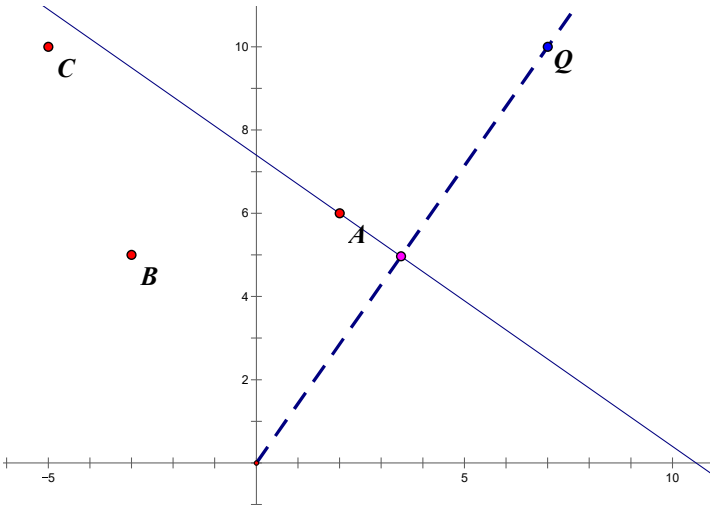
首先我们考虑一下暴力的写法，初看之下这道题是一道数据结构题，数据结构题的暴力一般都比较好写，只要按照题意模拟即可。对于询问区间$[l,r]$，我们可以简单粗暴的枚举$[l,r]$内的所有元素S_i，并统计$(x,y)\times S_i$的最大值即可。

分析这个算法的复杂度，**由于n,m同级，故在复杂度计算时不再区分**，询问次数是$O(n)$的，而区间长度也可以达到$O(n)$级别，所以暴力算法的时间复杂度$O(n^2)$，期望得分20分。

5.2 观察答案的性质

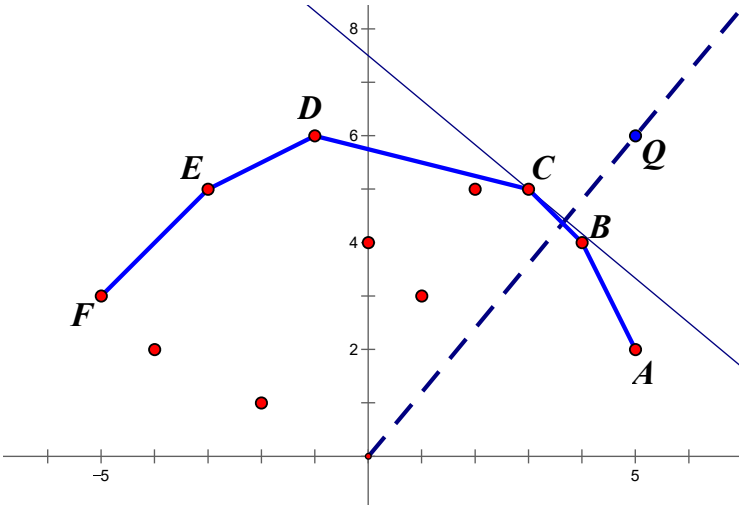
点积较叉积更为简单，我们不妨把叉积写成点积的形式，令$S_i=(x_i,y_i)$，对于询问x,y，有$(x,y)\times(x_i,y_i)=(-y,x)\cdot(x_i,y_i)$。把$(x,y)$变换成$(-y,x)$，则问题变为求出$(x,y)\cdot(x_i,y_i)$的最大值，之后的讨论将不再使用叉积。

由于$\overrightarrow{OA}\cdot\overrightarrow{OB}=\left|\overrightarrow{OA}\right|\cdot\left|\overrightarrow{OB}\right|\cos\angle AOB$，答案与$\overrightarrow{OB}$在$\overrightarrow{OA}$方向上的投影长度成正比，故点积最大值可以考虑这样来求，使用一条与$\overrightarrow{OQ}$垂直的直线，从无穷远处向原点O扫过来，碰到的第一个点即为答案。



例如上图，直线第一个扫到的点为A，则答案就是$\overrightarrow{OQ}\cdot\overrightarrow{OA}$。

继续观察$\overrightarrow{OQ},\overrightarrow{OS_i}$的角度取值，发现它们的角度(arctan)都是在$(0,\pi)$区间内，容易发现答案一定在区间内元素的上凸壳上。



例如上图，询问$\overrightarrow{OQ}$的答案就是$\overrightarrow{OC}$。

如果我们可以快速求得询问区间的上凸壳，那么在凸壳上二分斜率可以做到单次询问$O(\log n)$的复杂度。

于是问题的关键变成了如何维护凸壳。

5.3 以凸壳为代表的一类区间信息的维护

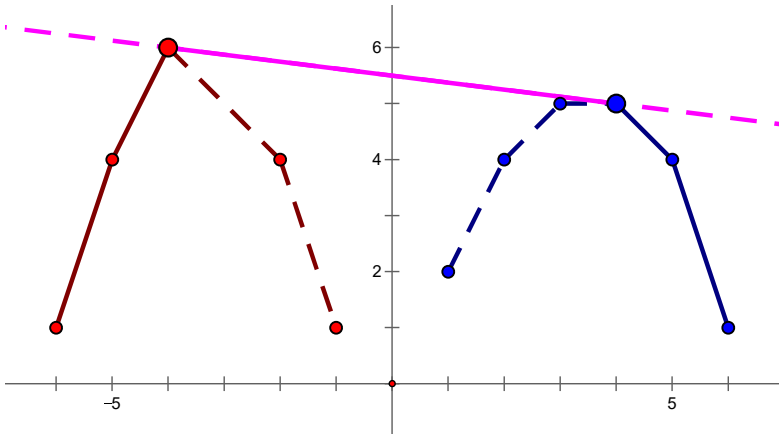
在维护区间信息时，有一些信息本身的特性是需要关注的，并且直接影响了我们维护区间信息的方式与难度。

5.3.1 是否支持快速合并

对于两个区间$[l, mid]$和$[mid+1, r]$，若这两个区间的信息能在$O(1)$或$O\left(poly\left(\log(r-l)\right)\right)$的时间复杂度内合并，我们就称这种信息支持快速合并。

支持快速合并的例子：

- 区间最值（合并复杂度$O(1)$）
- x关于下标单调的凸壳（使用平衡树维护并二分斜率是可以做到在$O\left(\log^2 n\right)$的时间复杂度内合并的，如下图所示）



不支持快速合并的例子：

- 区间点集的凸壳
- 区间数集的mex[2]

支持快速合并的信息都可以使用线段树等分治类区间数据结构进行维护，例如线段树维护区间最大/最小值已经是家喻户晓的技能了。

5.3.2 是否支持快速插入删除

对于一个区间$[l,r]$，若我们能在$O(1)$或$O\left(poly\left(\log(r-l)\right)\right)$的时间内求出$[l-1,r]$和$[l,r+1]$的值，则我们称这种信息支持快速插入。

同理若能在$O(1)$或$O\left(poly\left(\log(r-l)\right)\right)$的时间内求出$[l+1,r]$和$[l,r-1]$的值，称这种信息支持快速删除。

一个显而易见的结论就是：支持快速合并的信息一定支持快速插入（例如在右端插入可以看成合并区间$[l,r],[r+1,r+1]$）。

支持快速插入而不支持快速删除的例子：

- 区间点集的凸壳（使用平衡树等数据结构维护）
- 区间点集的虚树（使用平衡树维护相对dfs序）

支持快速删除而不支持快速插入的例子：

[2]mex:集合中未出现的最小自然数

- mex

同时支持快速插入删除的例子：

- 区间和的常数次多项式函数（如区间和的平方）
- 区间内不同的元素个数

支持区间快速插入删除的信息维护都可以使用莫队算法[3]来解决。由于莫队算法与本文的讨论核心关系不大，这里不展开叙述。

5.4 算法2：分块

由于凸壳本身不支持快速合并和快速删除，而本题是存在删除操作和区间询问的，于是传统的区间分治类数据结构如线段树是不能直接维护的，我们考虑分块。

将序列分为S块，每块大小为$\frac{n}{S}$，对每个块维护块内元素的凸壳。插入和删除的时候重构最后一块的凸壳，于是插入和删除的复杂度都是$O\left(\frac{n}{S}\right)$。

考虑询问区间$[l,r]$，我们需要枚举被区间$[l,r]$完全包含的区间，并在这些区间的凸壳上二分，复杂度是$O(S\log n)$的。

于是总复杂度就是$O\left(\frac{n^2}{S}+nS\log n\right)$，当取$S=\sqrt{\frac{n}{\log n}}$时，时间复杂度最小为$O\left(n\sqrt{n\log n}\right)$。

5.5 算法3：二进制分组

测试点4特殊性质：无2操作，3操作的询问为全部区间。

这个测试点中，我们只要维护全部元素的凸壳，并且只需要支持插入操作即可。

这个问题比较经典，可以使用平衡树维护凸壳。本文在这里介绍一种非传统的方法，采用二进制分组[4]来解决。

我们把序列分为连续的若干组，对每组维护它的凸壳，初始时序列为空。每插入第一个元素时，把这个元素单独分为一组，显然这一组的凸壳就是这个元素本身，如果末尾的两组大小相同，就把末尾的两组合并成一组。

[3]莫队算法：详见2010年集训队作业《小z的袜子》题解

[4]二进制分组：详见许昊然2013年国家集训队论文《浅谈数据结构题的几个非经典解法》

容易证明，这样的分组方式会使得每一组的大小都是2的次幂，并且组的大小严格递减，从而任意时刻组数不会超过$O(\log n)$。

考虑合并两个大小相等的组（不妨设大小为m），由于两个点集并的凸壳一定在两个点集的凸壳的并上，我们可以直接按照x坐标为关键字合并两个组的凸壳，再用Graham算法$O(m)$的计算凸壳即可。

下面分析合并过程的时间复杂度，考虑每个元素对于复杂度的贡献，一个元素被合并一次，所在组的大小一定会翻倍，所以每个元素对复杂度的贡献都不会超过$O(\log n)$，于是复杂度就是$O(n \log n)$。

注意到询问操作需要在每个组内的凸壳上都进行一次二分，所以复杂度是$O\left(n \log^2 n\right)$的，于是这个算法的总复杂度就是$O\left(n \log^2 n\right)$的。

5.6 算法4：时间倒流

测试点5特殊性质：所有2操作在1操作的后面，3操作的询问为全部区间。

二进制分组只涉及到新建与合并，是不能处理删除操作的，观察这个数据类型的性质，所有的删除都在插入后面，这提示我们可以把操作序列分成两半处理。

于是，对于前一半，测试点4的二进制分组做法，对于后一半，我们把时间倒流，这样删除操作就变成了插入操作，依旧套用二进制分组的做法。

时间复杂度为$O\left(n \log^2 n\right)$，把序列分成两部分的想法提示我们往分治的方向思考。

5.7 算法5：线段树维护静态序列

测试点6特殊性质：所有3操作都在1操作和2操作后面。

把所有插入删除操作都处理完之后，询问是在一个静态序列S上的，对于静态序列上的区间询问，我们可以使用经典的分治区间数据结构线段树。

对线段树上每个区间$[a,b]$我们维护该区间内所有元素的凸壳，再建树的时候我们可以直接采用测试点4中提到的线性合并凸壳的做法。这部分的复杂度是$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$。

接下来考虑查询区间$[l,r]$，一个区间能拆成$O(\log n)$个线段树节点所代表的区间的并，我们在这些节点上存储的凸壳上二分即可，时间复杂度是$O\left(n \log^2 n\right)$。

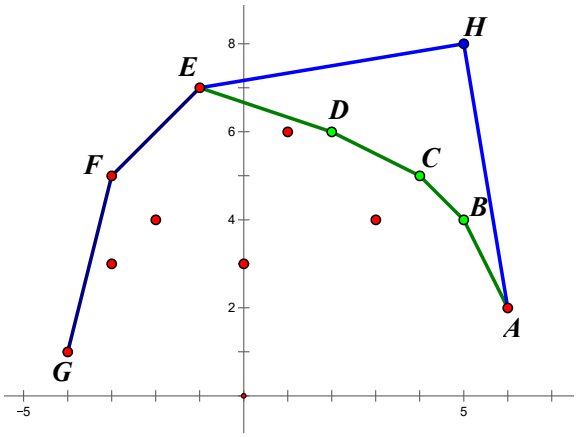
5.8 算法6：在操作树上分治

测试点7特殊性质：对于所有3操作有$l = 1$，内存限制为128M。

5.8.1 数据结构的做法

本题不要求强制在线，那么对于一个询问$[l, r]$，一定存在某个时刻序列长度恰好为r且与询问时序列相同，我们把询问记录在这个时刻上。又因为$l = 1$，所以相当于每次询问全体序列，我们只要维护所有元素的凸壳即可。

考虑使用伸展树(Splay)进行维护，注意到插入一个点时从凸壳中替换下来的点在原凸壳上一定形成一个区间，如果在节点上记录斜率，我们就可以二分出区间的左右端点，然后在Splay上把这一段分割下来并保存，在删除末尾元素的时候，把分割下来的区间再合并回去即可。



例如上图，H点加入之后把B, C, D三个点替换下来并保存。

由于记录了斜率，询问可以直接在Splay上二分，这样所有操作的复杂度都是$O(n \log n)$。

5.8.2 序列上的做法

插入和删除操作考虑起来有一定的困难，我们不妨先考虑序列为静态的情况，每次询问序列为一个前缀$[1, p]$。

我们可以采用分治做法，考虑处理一个区间$[l, r]$，令$mid = \frac{l+r}{2}$，我们把所有$p \geq mid$的询问拿出来，注意这些询问都是覆盖了$[l, mid]$区间的，我们一次性

处理$[l, mid]$区间中的元素对p在$[mid, r]$区间里的询问的贡献。处理完贡献之后，我们按照$p < mid$和$p \leq mid$把询问划分成两部分，分别往左边和右边递归分治计算。当分治到区间内只有一个元素的时候，直接用这个元素更新这个点上的所有询问即可。

下面就是考虑如何快速处理$[l, mid]$内元素对$[mid, r]$区间内询问的贡献：

方法A:我们对$[l, mid]$区间内元素建立出凸壳，然后枚举右边每个询问，在左边已经建立好的凸壳上二分即可，这样一次计算贡献复杂度是$O(n \log n)$的。

方法B:同样先对$[l, mid]$区间的元素建立出凸壳，然后把$[mid, r]$区间里的询问按照极角序排序，从前往后处理询问，凸壳上最优点的移动是单调的。所以我们只要设两个指针i, j分别在询问序列与凸壳上扫描即可，下面给出这部分的伪代码。

Algorithm 1 用两个指针i, j在排好序的询问和凸壳上扫描更新答案

```
for i = 0 to Q.size − 1 do
    while j + 1 < H.size and H_{j+1} · Q_i > H_j · Q_i do
        j = j + 1
    end while
    ans_i = max(ans_i, H_j · Q_i)
end for
```

一次扫描的时间复杂度为$O(n)$，由于要排序，实际复杂度是$O(n \log n)$。注意到排序是可以在分治的同时使用归并排序完成的，所以总复杂度$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$。

分治算法在静态的序列上复杂度并不比平衡树维护凸壳优秀，但是却有更良好的推广性。

5.8.3 操作树上的推广

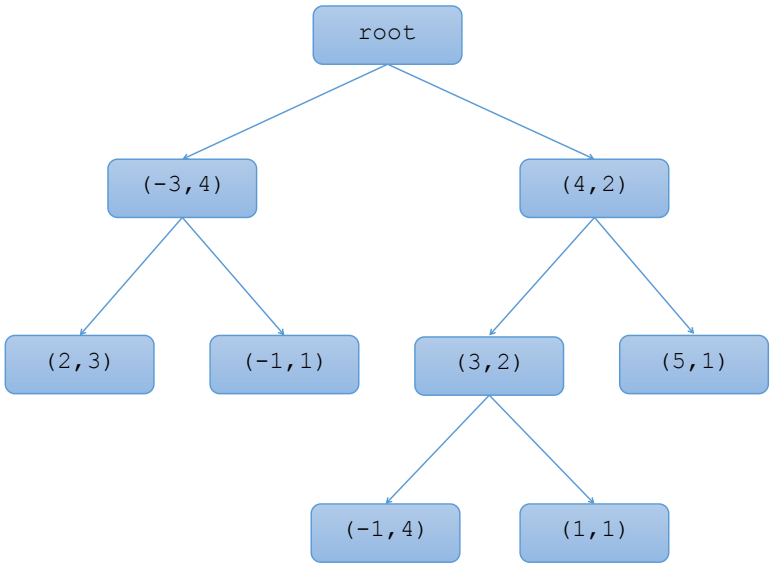
观察数据结构做法，发现很像在一棵树上进行dfs的过程。既然问题可以离线，我们干脆把这棵操作树建立出来。操作树建立方法如下：

1. 初始时建立一个根$root$，令指针$p = root$

2. 对于插入操作，新建一个点x，令$father_x = p$，再令$p = x$

3. 对于删除操作，令$p = father_p$

下图就是其右侧数据建立出的操作树




```
14
1 -3 4
1 2 3
2
1 -1 1
2
2
1 4 2
1 3 2
1 -1 4
2
1 1 1
2
2
1 5 1
```

由于只在插入时新建节点，这样建立出的操作树节点数是不超过插入操作的个数的。接着我们可以发现询问区间是树上一条自上而下的链，在本测试点中，链的顶端是$root$，我们把询问链为$root \to p$的询问记录在节点p上。

我们考虑推广分治算法，在操作树上，我们可以直接使用点分治。方便起见我们在点分治中定义一些记号，对于树上一个连通块，定义其中深度最浅的节点作为连通块的根，记为R，重心记为G。

我们在分治结构的每一层一次性处理链$R \to G$上的元素对G子树内的所有询问的贡献，这可以直接套用上文说的方法A或方法B。因为树分治结构不容易使用归并算法优化，所以方法A和方法B一次计算贡献的复杂度都是$O(n \log n)$。

由于重心把树分成了至少两个大小不超过$\frac{n}{2}$的连通块，于是总复杂度$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) = O\left(n \log^2 n\right)$。

5.9　算法7（满分做法A）

通过思考解决$l = 1$的情况（测试点7），我们拿到了一枚利器——树分治，

接下来我们尝试去解决原问题。

首先套用算法6，建立出操作树，然后在操作树上点分治，唯一不同的是计算链$R \to G$上的元素对G子树内的所有询问的贡献。因为$l = 1$的条件并不满足，所以G子树内的询问的区间不一定完整包含链$R \to G$，而是这条链的一个后缀。

我们把这个问题拿出来，发现这个问题与5.8.2中提到的在序列上询问前缀是等价的。于是我们把链$R \to G$取出并翻转，将询问加入之后采用5.8.2的做法，选用归并优化的做法B，可以做到一次计算$O(n \log n)$的复杂度。

于是这个算法的时间复杂度就是$O\left(n \log^2 n\right)$。

树分治和归并算法的空间复杂度都是$O(n)$，所以空间复杂度是$O(n)$，可以拿到100分。

5.10　算法8（满分做法B）

观察问题的本质，把插入删除操作建立成一棵操作树T之后，询问区间的本质就是树上自上而下的路径。

我们仍旧考虑点分治，一次性处理所有过重心G的所有询问。首先把询问链$u \to G \to v$，拆成两条询问链$G \to u, G \to v$。观察这个问题，发现如果我们令G为根，则与测试点7中的情况完全相同，直接使用5.8.1中的数据结构做法可以得到$O(n \log n)$的复杂度。

于是总复杂度就是$O\left(n \log^2 n\right)$，树分治和Splay的空间复杂度都是$O(n)$，于是空间复杂度也是$O(n)$的，可以拿到100 分。

5.11　算法9（满分做法C）

树分治部分和算法7相同，在计算链$R \to G$上的元素对G子树内的所有询问的贡献时，使用数据结构维护。从G开始到R向上依次添加节点，使用数据结构维护加入的节点的凸壳。对于每个询问后缀，我们在加入到这个位置的时候在数据结构上二分出答案即可。

由于这里的数据结构只需要支持加点操作，而加点时暴力维护凸壳是能保证复杂度的，所以大多数平衡树或者直接使用set即可完成。

时间复杂度$O\left(n \log^2 n\right)$，可以拿到100分。

5.12 算法10：树链剖分

在5.8.3中我们已经把询问区间转化为树上的链，大多数关于树链的问题，除了点分治之外，还可以采用树链剖分来解决。树链剖分算法本身已经较为普及，这里不再赘述。

对每个询问链，我们使用树链剖分结构将其分成$O(\log n)$个dfs序中的区间。观察剖分出来的这些区间的性质，可以发现除了深度最浅的那个区间之外，所有的区间都是某条重链的前缀。

我们对dfs序按照算法5建立线段树，然后把不是重链前缀的区间在线段树上查询更新。接下来考虑所有重链，对于每条重链，自上而下插入节点，并使用数据结构进行维护。

同样只要支持插入即可，所以大多数平衡树或者set都可以完成。

考虑这个算法的时间复杂度，建立线段树是$O(n\log n)$的，对每个询问不是重链前缀的部分在线段树上查询是$O\left(n\log^2 n\right)$的，扫描每条重链并维护凸壳，使用Splay的话复杂度是$O(n\log n)$的。由于一个询问被拆分成了$O(\log n)$个重链前缀，每个前缀都需要做一次二分，所以复杂度是$O\left(n\log^2 n\right)$的。于是总复杂度就是$O\left(n\log^2 n\right)$的。

注意到这个算法需要把每个询问剖分出的$O(n\log n)$条重链记录下来，于是空间复杂度是$O(n\log n)$的，可以拿到90分。

5.13 算法11：线段树分治

接着考虑算法10，一个询问被拆分成$O(\log n)$个dfs序区间。而序列上的区间修改或询问是可以使用线段树**离线**处理的，这种方法叫线段树分治。我们首先来看一个例子：

5.13.1 例：向量[5]

你要维护一个向量集合，支持以下操作：

1. 插入一个向量(x,y)
2. 删除插入的第i个向量
3. 查询当前集合与(x,y)点积的最大值是多少。如果当前是空集输出0

[5]**题目来源：NOI2015模拟题by 杨定澄**

允许离线。

100%的数据：$n \le 200000, 1 \le x, y \le 10^6$

题解

对于每个向量，我们都能预处理出它加入集合的时间与从集合中删去的时间，那么每个向量在时间轴上就是一个区间。

考虑对时间轴建立线段树，把每个向量插入线段树可以拆分成线段树上$O(\log n)$个节点。

接下来对每个点建立凸壳，则对于每个询问，自上而下查询每个包含它的区间即可，这样的复杂度是$O\left(n\log^2 n\right)$。

考虑进行一些优化，我们把询问按照极角排序，接着用5.8.2中讲的做法线性求得答案，于是瓶颈在于排序。

事实上，如果我们把所有的向量按照x坐标排序后插入，把所有的询问按照极角排序后插入，那么在线段树的节点上就不用排序了，于是总复杂度就变成了$O(n\log n + Q\log Q + Q\log n)$。

5.13.2　本题中的线段树分治

注意到“向量”与本题有一些区别，“向量”中的询问是单点，而集合中的向量覆盖的是区间，本题中我们询问的是向量序列的区间信息。

我们依然可以沿用线段树分治的做法，首先把询问按照极角序插入线段树，然后在线段树上dfs，对每个线段树节点使用归并求出该节点所代表区间的凸壳，接着仍然使用5.8.2中提到的算法更新答案即可。

这样复杂度是$O(n\log n + Q\log n)$，由于我们使用树链剖分把一个询问拆成了$O(\log n)$个，于是$Q = O(n\log n)$，总复杂度就是$O\left(n\log^2 n\right)$。

由于要在线段树上保存$O\left(n\log^2 n\right)$个询问，故空间复杂度是$O\left(n\log^2 n\right)$的，可以拿到70-90 分。

5.14　算法12：再探二进制分组

算法6-11都要求允许离线，那么有没有一个可以支持在线的做法呢？

在算法3中我们引入了一种在线处理插入的方式——二进制分组，考虑对这个算法进行改进。

我们建立$t+1$层分组结构，t为最小满足$2^t \geq n$的整数（n是当前区间长度），第i层$(i \leq i \leq t)$的组大小为2^i，有$siz_i = \left\lceil \frac{n}{2^i} \right\rceil$组。注意到第$i$层的每一组都在第$i-1$层上被均分成两个等长区间，所以这是一个线段树结构，于是可以按照线段树定义左右孩子和父亲的关系。

如果只有插入还是一样的做法，首先在第0层添加一组，接着从第$i=1$层开始，如果第i层可以增加一组，那么将其左右孩子组合并得到新添加的一组。

删除操作不太好处理，因为删除一个元素可能导致前若干层的最后一个组信息错误，而每次暴力重构的复杂度是$O(n)$，这是不可接受的。我们引入替罪羊树的思想，当一个组信息出现错误的时候，我们不立即重构更新，而是给这个组打上标记，查询到这个组的时候我们递归它的左右孩子，直到碰到没打标记的组才在组内二分计算答案。但是也不能一直不更新，否则查询时递归到底层节点数会达到$O(n)$个，这也是不可接受的。于是考虑这样一种更新机制：

对于每层，我们至多允许最后一个组是有标记的。

这样，当第i层需要增加一组的时候，我们才把原来的最后一组暴力更新一下；删除的时候只要自下而上打标记即可。

接下来计算这个算法的时间复杂度。

5.14.1 重构更新的复杂度

对于第i层，我们统计两种事件：

1. 最后一个组被打标记，复杂度$O(1)$。
2. 重构最后一个组并添加带有标记的新的一组，复杂度$O\left(2^i\right)$。

令len_i表示第i层没有标记的组的总长度，令$E_i = n - len_i$，那么事件1是相当于E_i从0变为2^i，事件2是相当于E_i从$2^i * 2$变为2^i。而一次插入或删除操作对于E_i的改变幅度是± 1。

令势能函数$\delta(i) = \left|E_i - 2^i\right|$，则一次插入或删除操作对$\delta(i)$的贡献至多为1，而一次事件对$\delta(i)$的贡献为$-2^i$。于是两次事件间的操作数至少是$2^i$的，于是均摊复杂度是$O(n)$的。

考虑到有t层，于是重构更新的总复杂度就是$O(tn)$。

5.14.2 查询的复杂度

之前提到这是一个线段树结构，那么首先把询问区间$[l,r]$拆成$O(\log n)$个组。对于每个组，有三种情况，考虑每一种情况的贡献：

1. 没有被打标记。

   对于这种情况，直接查询这个组即可。

2. 有标记而且左右孩子都没有被打标记。

   对于这种情况，我们会在左右孩子中分别查询，仅仅是常数乘以2。

3. 有标记而且右孩子上也有标记。

对于这种情况，我们在左孩子中查询之后递归到右孩子。注意到这里的左孩子一定是某一层的倒数第二个组，这样的组只有$O(\log n)$个。

于是我们查询的区间总数是$O(\log n)$的，于是处理一次询问操作的复杂度就是$O\left(\log^2 n\right)$。那么总复杂度就是$O\left(tn + n\log^2 n\right) = O\left(n\log^2 n\right)$。由于要维护$t$层，所以空间复杂度是$O(tn) = O(n\log n)$的，可以拿到90分。

6 思路小结

本题比较复杂，直接思考起来较为困难，大致分为几个步骤。

首先分析出需要维护的信息，本题中我们需要维护询问集合的凸壳，凸壳是典型的不易快速合并的信息。

接着考虑部分分，并从中获得一些提示。

只有插入且询问全局的部分可以使用二进制分组在线完成。

没有修改的部分可以采用线段树完成。

$l = 1$的部分是比较关键的部分，而且难度较大。我们考虑了序列上的情况，给出了分治的做法。接着引入了操作树的概念，并把分治算法推广到树上即点分治算法。

拿到点分治这一门武器之后，我们尝试解决原问题，观察原问题与$l = 1$的部分不同的地方在于子树内的询问询问不一定是根到重心的链，可能是这条链的一个后缀。继续观察发现这个问题就是$l = 1$的部分在序列上的问题，继续分

治解决。于是我们得到了一个时间复杂度$O\left(n\log^2 n\right)$，空间复杂度$O(n)$的优秀算法。

可见，虽然标题是“我们仍未知道那天所看见的数据结构的名字”，但是我给出的满分算法并没有用到数据结构，而是采用了两次分治简化问题。

其实本质就是把CDQ分治[6]的思想放到树上，序列分治变成点分治，观察到每层分治结构上的贡献统计是序列上的特殊子问题并继续套用分治是求解本题的关键。

以上算法要求离线，考虑强制在线加强版本，空间要求放宽到$O(n\log n)$。

这使得我们回想起只有插入的部分分是支持强制在线的，于是把二进制分组改进成线段树结构，并引入替罪羊树思想。

最后我们设定每一层至多只有最后一个区间被打了标记，并分析出了维护信息的复杂度是$O(n\log n)$的，瓶颈在于二分，于是总复杂度就是$O\left(n\log^2 n\right)$。

应该说这是这一类问题较为通用的在线算法，其空间复杂度是$O(n\log n)$。

本题到此已经完美的解决了，但本题的意义不止于此，在解决本题时用到的思想和方法都有很大的实用与推广价值。

7　简单推广

7.1　双端插入删除

以上的讨论范围都是在末尾插入删除元素的问题。其实我们可以非常容易的推广到两端插入删除的问题上来。

7.1.1　算法7的推广

对于算法7，我们考虑改进操作树的构建方法：

1. 初始时建立一个根$root$，令指针$p = q = root$
2. 对于在开头插入操作，新建一个点x，令$father_x = p$，再令$p = x$
3. 对于在末尾插入操作，新建一个点x，令$father_x = q$，再令$q = x$

[6]CDQ分治：详见陈丹琦2008年国家集训队论文《从Cash谈一类分治算法的应用》

4. 对于在开头删除操作，令$p = father_p$

5. 对于在末尾删除操作，令$q = father_p$

对于每个询问区间$[l, r]$，其在树上对应的链就是链$p \to q$上第l个节点到第r个节点组成的链。但是我们并不能直接使用算法7处理，因为算法7要求询问链必须是树上自上而下的区间，而改进之后的操作树并不能满足这一点。

因此需要改进，只要把一条询问链$u \to v$拆成$w \to u$和$w \to v$两条链即可，其中w是u, v的最近公共祖先(LCA)。

同样算法9也可以推广到改进后的操作树上来，算法8由于是处理过根的询问，所以并不需要把询问拆成两条自上而下的链，直接处理即可。详细的扩展方法与算法7类似，这里不再赘述。

7.1.2 算法12的推广

二进制分组只能支持在一端操作，我们考虑算法12中改进的二进制分组，其实这是一个线段树结构，那我们可以直接使用线段树来代替。

初始时建立长度为$2n$的线段树，并把初始位置放在正中间。我们把更新机制改为每层最多允许两个组有标记（左右各一个），则插入删除时直接沿用算法12的处理方法。所有的复杂度分析在这里依然是成立的（我们只要把左端和右端的δ分开计算即可），只是询问的时候，由于两边两边的倒数第二组可能都被访问，所以常数会乘以2。

7.2 维护其他不能快速合并的区间信息

事实上，在信息不易快速合并时，要求支持末尾插入删除和区间询问的问题大都可以使用本文阐述的算法来完成。我们来看一个例子：

7.2.1 例：吉利树[7]

给一棵n个点的带边权的树，你需要维护一个序列，支持三种操作：

1.在末尾插入一个点x

2.删除末尾的点

[7]**题目来源：**by吉如一

3.询问区间$[l,r]$中离x最远的点到x的距离。

操作数为m,n,$m \leq 100000$。

Subtask1：允许离线

Subtask2：强制在线

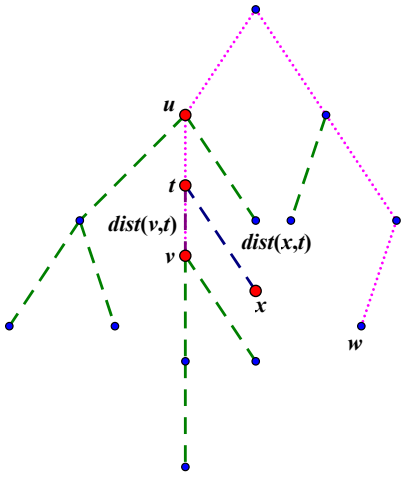
考虑需要维护的信息

若给定一棵树，如何求距x最远的点？我们可以采用树形DP预处理出树上每个点的最远点，这样时间复杂度是$O(n)-O(1)$[8]的。

若给定一棵树和一个树上节点的集合S，求x到S中最远点的距离该如何解决呢？我们依旧采用树形DP预处理，时间复杂度仍然是$O(n)-O(1)$。当$|S|$很小的时候这个算法的复杂度是不够优秀的，考虑使用虚树算法[9]。虚树算法在信息学竞赛中已经较为普及，这里不展开叙述。我们知道建立虚树的时间复杂度是$O(|S|\log n)$的，接着我们在虚树上树形DP预处理出每个虚树上点u到点集中的最远点距离d_u，对于每个询问点x，我们在dfs序上二分求出x到虚树上最近的两个点u,v，显然u,v必为虚树上的一条树边，设u是v的父亲，令$t=LCA(v,x)$。

若x的最远点在s一侧$(s=u\,or\,v)$，则答案为$dist(x,t)+d_s-dist(s,t)$，故$ans(x)=\max(dist(x,t)+d_u-dist(u,t),dist(x,t)+d_v-dist(v,t))$。

下面看一个图例理解一下：



如上图所示，$d_v=dist(v,w)$，$ans(x)=dist(x,w)=dist(x,t)+d_v-dist(v,t)$。

[8] $O(n)-O(1)$：即预处理$O(n)$，每次询问$O(1)$

[9] 虚树算法：http://www.lxway.com/40452412.htm

Subtask1题解

首先建立出操作树将询问区间变成树上的一条链。

如果使用算法7点分治套分治，那么需要解决的就是区间信息合并的问题。

虚树的构建是基于dfs序的，那么我们只要维护区间元素按dfs序排序的结果，那么有序表的合并是$O(n)$的，由于建立虚树的复杂度瓶颈是求LCA，而LCA是可以使用欧拉序列+RMQ优化为单次$O(1)$的，于是一次合并的总复杂度就是$O(n\log n)$。

这样内层分治的复杂度是$O(n\log n)$，故外层分治的复杂度为$O\left(n\log^2 n\right)$，可以解决这个子任务。

考虑另一种方法例如算法8，那么要解决插入元素x的问题。

我们采用平衡树按照dfs序维护虚树节点，那么插入元素时在平衡树内二分找到离x最近的边从而找出虚树树边上离x最近的点y，把x和y都插入集合中。那么每次插入元素只要在平衡树中插入至多两个节点，对于虚树边的修改次数也是$O(1)$的，复杂度是$O(\log n)$，同时我们只要记录修改就能方便的支持回退了。

算法9的解法与算法8类似，这里不再叙述。

Subtask2题解

算法12只需要合并区间，而区间信息的合并在7.2.1中已经可以做到$O(n)$的复杂度。于是我们直接把算法12维护的区间信息改为虚树就可以解决本题了，复杂度的分析和Unknown一题完全一样，重构更新均摊$O(n\log n)$，查询$O\left(n\log^2 n\right)$可以解决本题。

8　命题思路与总结

8.1　Unknown命题思路

近年以来出现了很多要求维护区间信息的题目，这类题目大多使用区间数据结构维护。相对不容易维护的信息中以凸壳出现最为频繁，需要通过对时间分治，二进制分组等非传统方法解决的题目也越来越多。

NOI2014第二天第三题“购票”是一道典型的需要维护凸壳并支持区间查询来优化DP的题目，吕凯风学长现场使用树分治算法AC了此题，并在不久之后

把这个思想分享给了我。这个思想对我很有启发，并激发起了我对这一类信息维护方法研究的兴趣。

我研究这一类算法并命制的集训队互测题“我们仍未知道那天所看见的数据结构的名字”是一道有难度的题目，其部分分做法涵盖了多个非传统方法求解传统数据结构题的分支，例如二进制分组，时间倒流，线段树分治等。而得到其满分做法需要选手有敏锐的观察力，严谨的分析能力以及对分治算法的深刻理解和熟练掌握。同时这题对代码能力的要求也比较高，预计选手可能要花上2至4个小时才能拿到本题的全部分数。

8.2 方法总结

本文的最后总结出这一类在端点处插入删除并维护不易快速合并的区间信息的通用思考过程与解决方法。

如果允许离线，建立出操作树并考虑分治，对于能$O(n)$合并区间的可以继续套用分治（算法7）；否则观察信息是否同时支持加入和删除元素，可以的话使用数据结构从重心开始dfs统计（算法8）；如果只能支持快速插入和快速删除中的一个，可以按照某个顺序扫描根到重心的链（算法9）。

以上算法的时间复杂度都是$O\left(n\log^2 n\right)$，空间复杂度都是$O(n)$。

如果强制在线，那么考虑引入替罪羊树思想的二进制分组（算法12）。在复杂度分析中，复杂度瓶颈不在于重构更新，而在于询问。于是我们可以接受$O(n\log n)$的重构复杂度，时间复杂度仍然是$O\left(n\log^2 n\right)$的。对于有的题目，我们可以采取更优的查询方法（例如归并定位），将询问复杂度优化到$O(\log n)$，从而把总复杂度降为$O(n\log n)$

由于要建立$O(\log n)$层分组结构，空间复杂度是$O(n\log n)$ 的。

Unknown只是诸多此类问题的一个代表，存在许多种不同的解法，这些解法各有各的长处并都具有一定的实用性，所以我觉得激发选手思考这方面算法的意义甚至大于题目本身。希望我的这道题起到一个抛砖引玉的作用，能引起大家对于这一类问题的兴趣，并提出一些更有效，更方便的算法。如果在此方面的研究有所收获欢迎大家来找我讨论。

9 感谢

- 感谢叶国平老师在学习生活上的指导和关心。
- 感谢CCF提供交流的平台与机会。
- 感谢吕凯风学长提供命题灵感。
- 感谢吉如一同学，杨家齐同学，吴作凡同学，毛啸同学给予的帮助。
- 感谢吴作凡同学，吉如一同学，倪星宇同学，王蕴韵同学为本文审稿。

参考文献

[1] SDOI2014 向量集 题解 http://www.cnblogs.com/joyouth/p/5350714.html

[2] 许昊然,《浅谈数据结构题的几个非经典解法》,2013年信息奥林匹克中国国家队候选队员论文

[3] 陈丹琦,《从Cash谈一类分治算法的应用》