可持久化数据结构

陈立杰

杭州外国语学校

2012 年 5 月 23 日

所谓的持久化数据结构,就是保存这个数据结构的所有历史版本,同时用它们之间的共用数据减少时间和空间的消耗.

很多数据结构都可以可持久化,比如线段树,平衡树,块状链表.

可持久化线段树Q&A

如何实现？实现不了你讲个**.

可持久化线段树Q&A

如何实现？实现不了你讲个**.

有什么用处？没有用处你讲个**.

可持久化线段树Q&A

如何实现？实现不了你讲个**.

有什么用处？没有用处你讲个**.

代码复杂度？写不出来你讲个**.

可持久化线段树Q&A

如何实现？实现不了你讲个**.

有什么用处？没有用处你讲个**.

代码复杂度？写不出来你讲个**.

时间复杂度？跑不出来你讲个**.

可持久化线段树Q&A

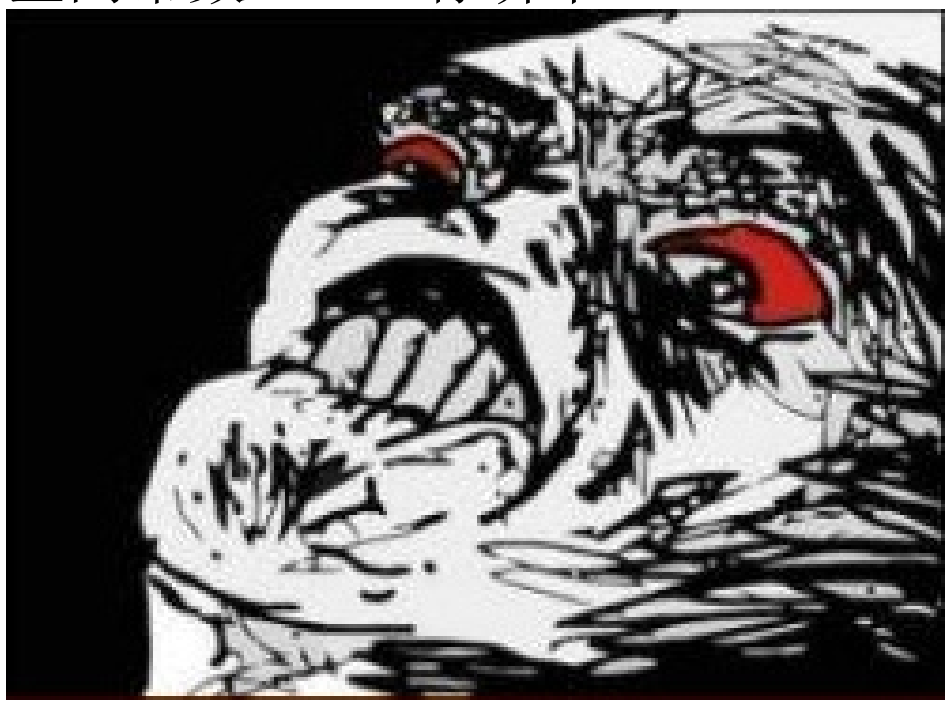
如何实现？实现不了你讲个**.

有什么用处？没有用处你讲个**.

代码复杂度？写不出来你讲个**.

时间复杂度？跑不出来你讲个**.

空间常数？MLE你讲个**.



线段树的一些记号

若t表示一个节点.

$\text{left}(t), \text{right}(t)$分别表示$t$的左右孩子.

若t的范围是$[l, r)$,那么$\text{left}(t)$的范围是$[l, \frac{l+r}{2})$,$\text{right}(t)$的范围是$[\frac{l+r}{2}, r)$.

$\text{f}(t)$表示在t上记录的一些信息,比如$\text{cnt}(t)$是t中数的个数之类.

指针实现的线段树

一个节点表现为一个结构struct或者class在C++,record在Pascal.

一个节点维护两个指针指向它的左孩子和右孩子.

操作都递归进行.

可持久化

我们想让线段树支持一些操作,同时能够维护所有的历史版本.

也就是说,每次操作返回一个新的线段树!它是这个操作的结果线段树.

比如不妨考虑支持单点修改的最基础的线段树.

假设一开始数列是$T_1 = [1, 2, 3, 4, 5]$

我们修改第3个位置变成10,那么得到$T_2 = [1, 2, 10, 4, 5]$.

然后我们仍然可以对T_1进行询问和操作!是不是很神奇啊.

感性的考虑一下,一次修改操作更改的节点只有$O(\log n)$个,只重建这些点并且尽量重用其它的点,就是我们的目的.

让我们来考虑如何实现可持久化线段树.

可持久化线段树的实现:单点修改

首先我们的目标是,修改一个位置上的值,同时不影响所有目前保存的历史版本的运作,得到一个新的版本的线段树.

由于线段树的实现是递归的,那么我们也来递归的看待这个问题.

首先考虑一个叶子节点,那么我们只需要新建一个新的叶子节点,它的值是修改过后的值,那么就能得到一个当前版本.

再考虑一个非叶子节点,由于它的两个孩子中最多只会有一个被修改,不妨看成左孩子,那么我们对左孩子递归调用函数,得到左孩子的修改后版本,然后将当前节点拷贝一份,右孩子不变,左孩子为修改后的版本,就能得到当前节点修改后的版本.

注意到会被修改的节点,只能是被修改的叶子节点的祖先,故每次只需要新建$O(\log n)$个结点.同时我们只是在新建节点,没有对任何节点的信息做修改,故历史版本也没有受到影响.

可持久化线段树的实现:询问

跟普通线段树一样递归询问即可.

可持久化线段树的实现:标记

线段树中有一个非常经典的操作:打标记.

让我们考虑如何让可持久化线段树实现标记.

当我们访问到一个点时,如果它上面有标记,我们就将它的标记下传.注意到这里我们对点修改了,但这是没有问题的,因为下传标记后,该点还是和原来的点等价,不会影响之前的历史版本.

注意到下传标记的时候,我们不能修改它的孩子的值,我们得新建2个节点表示打完标记之后它的孩子.

打标记则类似单点修改,如果该点被整个覆盖,就新建一个节点表示打完标记后的它,否则的话就新建一个点,没被覆盖的孩子不变,被覆盖的孩子替换为修改后的版本.

块状链表

块状链表,就是$O(\sqrt{n})$个用链表连起的数组.

保证相邻的两个块大小之和$\geq \sqrt{n}$,一个块的大小$\leq 2\sqrt{n}$.

插入元素:我们找到那个元素所在的块,在它内部插入那个元素,复杂度$O(\sqrt{n})$.

删除元素:我们找到那个元素所在的块,在它内部删除那个元素,复杂度$O(\sqrt{n})$.

维护:每次进行完操作之后,我们扫描一遍所有的块,如果有相邻两块的大小之和$< \sqrt{n}$,那么将它们合并,如果有一块大小$> 2\sqrt{n}$,那么将它分成两个尽量等大的部分.

复杂度:容易看出所有操作的复杂度都是$O(\sqrt{n})$.

可持久化

让我们考虑如何持久化块状链表.也就是维护块状链表的历史版本.

我们每次修改一个块的时候,改为新建一个值为修改过后的块的新块.

同时用一个大小$O(\sqrt{n})$的数组而不是链表保存所有块.

修改之后用一个新数组保存新的块序列.

注意到在块序列中我们使用指针.

区间第k大问题

经典问题,做法很多,是广大人民群众喜闻乐见的一种问题.

有很多变种,不过百变不离其宗.

问题回顾

让我们来回顾一下经典的区间第k大问题.

问题描述:

给n个数$a_0, a_1, ..., a_{n-1}$,每次询问$a_l, a_{l+1}, ..., a_r$中,第k大的数是多少.

回答方式有在线和离线2种,也有支不支持修改一个位置的数的操作的区别.

经典算法

我们来回顾一下这个经典问题的经典算法.

无修改在线

复杂度$O(A)+O(B)+O(C)$表示预处理复杂度$O(A)$,回答一次询问复杂度$O(B)$,空间复杂度$O(C)$

二分答案+线段树 $O(n\log n)+O(\log^3 n)+O(n\log n)$

二分答案+分块 $O(n\log n)+O(\sqrt{n\log n}\log n)+O(n)$

划分树 $O(n\log n)+O(\log n)+O(n\log n)$

按值建线段树 $O(n\log n)+O(\log^2 n)+O(n\log n)$

有修改在线

复杂度$O(A)+O(B)+O(C)+O(D)$表示预处理复杂度$O(A)$,回答一次询问复杂度$O(B)$,修改的复杂度$O(C)$,空间复杂度$O(D)$

二分答案+线段树套平衡树

$O(n\log n)+O(\log^3 n)+O(\log^2 n)+O(n\log n)$

二分答案+分块

$O(n\log n)+O(\sqrt{n\log n}\log n)+O(\sqrt{n\log n})+O(n)$

按值建线段树套平衡树

$O(n\log n)+O(\log^2 n)+O(\log^2 n)+O(n\log n)$

分块算法的要点是将每块大小设为$\sqrt{n\log n}$,可以降低复杂度.

可以看到划分树的复杂度十分优秀,但是这个结构较为复杂,代码复杂度比较高.

使用可持久化线段树的做法

主席树!ChairTree!

思想简单而有内涵,思路明晰而不乏深度,编码复杂度和思维复杂度非常低,代码量很小,时间复杂度最优.

群众们喜闻乐见!

使用可持久化线段树的做法

如何使用线段树找第k大:

找第k大跟第k小是等价的,为了方便我们找第k小.

这是一个经典的问题,我们假设数是在$[0, n)$之间,那么对于权值建立线段树. 每个节点t,用$\mathrm{cnt}(t)$表示节点内数的个数.

那么我们考虑当前在节点t,找节点t内部的第k小的数. 如果t的左孩子内部的数的个数$\geq k$,那么答案在左孩子内部,t跳到左孩子. 否则的话,答案就是右孩子内部的第$k - \mathrm{cnt}(\mathrm{left}(t))$个节点,$t$跳到右孩子即可.

这个的复杂度是$O(\log n)$.

同时增加一个数也只需要对线段树进行修改就可以实现.

权值线段树的运算

考虑两课结构相同的权值线段树a, b,权值线段树$a(+/-)b$的每个节点的cnt值是对应位置的a中节点的值$(+/-)$ 对应位置的b中节点的值.

考虑一棵权值线段树a和一个整数c,权值线段树$a \cdot c$的每个节点的cnt值是对应位置的a中节点的值乘c.

那么对于权值线段树$a - b$,我们想对它实行询问并不需要建出他,只需要维护a, b中在其对应位置的节点即可.

对于权值线段树$a \cdot c$,也是一样的道理.

同样的道理,我们考虑一个线段树

$$T = \sum_{i=1}^{k} a_i \cdot c_i$$

就是k棵线段树的代数和. 那么对线段树T进行询问,就需要$O(k \log n)$的时间.

无修改

考虑无修版

让我们先对所有数离散化,那么离散化之后,$a_i < n$.

接下来,我们用at_i表示$a_0, a_1, ..., a_{i-1}$这些数添加到上面说的权值线段树形成的线段树.

那么at_i可以通过at_{i-1}修改一个位置得到.

使用可持久化线段树得到所有的at_i只需要$O(n \log n)$的时间和空间.

那么我们考虑询问区间$a_l, a_{l+1}, ..., a_{r-1}$的第$k$大数.

注意到我们只需要在线段树$at_r - at_l$上找第k大就行了.

那么这个算法的复杂度就是$O(n \log n) + O(\log n) + O(n \log n)$

有修改

由于有修改了,我们想要得到at_i就比较难办了,因为一次修改会改掉$O(n)$个at_i的值.

但是我们可以用树状数组来维护at_i,就相当于维护一个权值线段树的前缀和.

那么at_i就可以表示成$O(\log n)$个权值线段树的和.

那么$at_r - at_l$自然可以表示成$O(\log n)$个权值线段树的带系数的代数和了.

那么就可以在$O(\log^2 n)$的时间内完成询问,同时修改时只需要修改树状数组中$O(\log n)$个节点,故修改复杂度也为$O(\log^2 n)$.

那么这个算法的复杂度就是$O(n \log n) + O(\log^2 n) + O(\log^2 n) + O(n \log^2 n)$

总结

使用可持久化线段树的算法时间复杂度无论有没有修改操作,都是已知算法中最优的.

同时这些算法非常易于理解和好写,非常不容易写错,在OI比赛中很适合使用.

(我会说划分树已经变成时代的眼泪了么)

令人遗憾的是带修改版本的空间复杂度较大,需要使用一些空间常数优化才能通过一些题目.

关于修改的一些细节

注意到我们的权值线段树,是需要离散化的,如果可以离线的回答问题,那么我们不妨先读入所有询问,然后进行离散化.

但如果必须在线呢?

实时开节点的权值线段树

让我们来打一个神奇的标记:"存在",一个节点的孩子一开始是不存在的,只有访问到它的时候,如果它上面有"存在"这个标机,就把这个标记推给它的孩子:让它不存在的孩子们变得"存在".

其实就是边访问边新建节点.

那么每次访问最多新建$O(\log n)$个节点.

那么我们直接对值域开值线段树,比如$[0, 2^{31})$,就可以解决在线修改的问题.

空间复杂度就很微妙了.

区间第k大问题EXT

标题中的EXT的意思就是EXTENDED,所谓的加强版.

既然我们的是持久化数据结构研究,那么我们应该也要支持在历史版本里询问第k大.

将询问改变成:询问第i次修改操作后,这个数列$a_l, a_{l+1}, ..., a_r$中,第k大的数是多少.

让我们来考虑怎么做,不妨用一个可持久化的线段树,这个线段树是对a中的下标位置开的,

范围为$[l, r)$的节点上保存一个包含$a_l, a_{l+1}, ..., a_{r-1}$的权值线段树.

那么$a_l, a_{l+1}, ..., a_r$就能被拆成$O(\log n)$个权值线段树的和.

询问就能在$O(\log^2 n)$的时间内完成.

同样,修改也可以在$O(\log^2 n)$的时间内完成并保存历史版本.

区间第k大问题EXTEXT

标题中的EXTEXT的意思就是EXTENDED's EXTENDED,所谓的加强版的加强版.

让我们再来支持一个操作:在一个位置插入一个数,同时还要支持历史询问.

这下问题的难度就大大上升了,如果可以离线的话,我们可以先得出所有数的最终位置再计算,可是如果必须在线的话,这个方法就不管用了.

我们的目标是将$a_l, a_{l+1}, \ldots, a_r$变成一些权值线段树的和.

不妨使用可持久化块状链表维护.

每个块我们维护一个权值线段树,表示这个块内部的数.为了支持在线修改我们需要实时开节点的权值线段树.

区间第k大问题EXTEXT

考虑询问$a_l, a_{l+1}, ..., a_r$,容易发现除了$O(\sqrt{n})$个数之外,其它的都属于某个块内部的权值线段树.

那么考虑一个询问,我们先将那$O(\sqrt{n})$个数建成权值线段树,需要$O(\sqrt{n}\log n)$的时间,然后在这个权值线段树和那些块内线段树的和中找第k 大,需要$O(\sqrt{n}\log n)$的时间,故总复杂度是$O(\sqrt{n}\log n)$

区间第k大问题EXTEXTEXT

标题中的EXTEXTEXT的意思就是EXTENDED's EXTENDED's EXTENDED,所谓的加强版的加强版的加强版.

让我们再来支持2个操作:删除一段数,复制一段数之后再在某一个位置插入这些数,同时还要支持历史询问.

同时数的数量不会超过10^5.

继续使用可持久化块状链表维护.

让我们考虑如何提取其中一段,显然一段由中间几个块,和头尾各$O(\sqrt{n})$个元素组成,我们将头尾那些元素建成新的块,然后再把中间那些块拿过来,就能得到提取的一段.

删除一段是同样的道理,删除几个块后,修改区间头尾的两个块即可.

区间第k大问题EXTEXTEXT

那么在某一个位置插入一段也是一样,我们插入之后扫描一遍进行维护即可.

注意到由于是可持久化的,修改都要通过新建一个块来完成.

那么这些操作的复杂度都是$O(\sqrt{n}\log n)$.

树上路径第k大问题

这类题目的经典问题是2008年CTSC的network.

群众们喜闻乐见的数据结构题.

问题回顾

问题描述:

给一棵边带权的树,每次询问$a \to b$这条路径上的边中,第k大的边权是多少.

同样有在线,离线,支不支持修改等区别.

经典算法

无修改+在线:

复杂度$O(A)+O(B)+O(C)$表示预处理复杂度$O(A)$,回答一次询问复杂度$O(B)$,空间复杂度$O(C)$

二分答案+树链剖分套线段树

$O(n\log n)+O(\log^4 n)+O(n\log n)$

按值建线段树 $O(n\log n)+O(\log^2 n)+O(n\log n)$

有修改+在线:

复杂度$O(A)+O(B)+O(C)+O(D)$表示预处理复杂度$O(A)$,回答一次询问复杂度$O(B)$,修改的复杂度$O(C)$,空间复杂度$O(D)$

二分答案+树链剖分套线段树套平衡树

$O(n\log n)+O(\log^4 n)+O(\log^2 n)+O(n\log n)$

按值建线段树套平衡树

$O(n\log n)+O(\log^2 n)+O(\log^2 n)+O(n\log n)$

注意到树链剖分的复杂度常数对一般的树都是非常小的.

使用可持久化线段树的算法

主席树!ChairTree!

思想简单而有内涵,思路明晰而不乏深度,编码复杂度和思维复杂度非常低,代码量很小,时间复杂度最优.

群众们喜闻乐见!

无修改

我们首先用$O(n\log n)-O(\log n)$的算法来回答$\mathrm{Lca}(u,v)$的询问,即两个点的最近公共祖先.

那么用at_u表示从根到点u的路径的这些边上的数组成的权值线段树.

at_u可以从它的父亲的at值修改一个数得到,故可以在$O(n\log n)$的时间内得到所有at值.

我们可以发现$u\to v$这条路径上的边组成的权值线段树$T=at_u+at_v-2\cdot at_{\mathrm{Lca}(u,v)}$.

那么只要在线段树T上找第k大即可.

复杂度$O(n\log n)+O(\log n)$.

有修改

让我们考虑修改之后怎么得到at_i.

我们先对树进行dfs得到dfs序.

那么修改一条边的权值,只会影响一颗子树的at_i,也就是dfs序中的一个区间.

那么我们用一个线段树来维护dfs序上的每个点的at_i即可.

那么at_i就能表示成$O(\log n)$个权值线段树的和.

复杂度$O(n \log^2 n) + O(\log^2 n) + O(\log^2 n)$.

同时,我们只需要让最外层的那个线段树持久化,就能回答历史版本的问题.

离线转在线

离线转在线.

空间换时间.

结合本来在不同时间轴的要素.

最近的给定权值的祖先

题目描述:

给一棵树,每个点都有一个权值,我们每次询问一个点往上,第一个权值为x的权值的编号.

必须在线.

算法

不妨令$a[u][c]$表示u往上第一个权值为c的点的编号.

那么考虑数组$a[u][*]$,注意到跟u的父亲f的数组$a[f][*]$,只修改了一个位置.

我们使用可持久化线段树维护数组$a[u]$,那么就能在$O(n \log n)$的时间和空间复杂度内得到所有$a[i][*]$.

复杂度$O(n \log n) + O(\log n)$

注意到如果改成可持久化块状链表,复杂度就变成.

$O(n\sqrt{n}) + O(1)$.

例题:Middle

让我们来练习一下.

题目大意

一个长度为n的序列a,它的中位数$\mathrm{middle}(a)$,定义为序列a,从小到大排序排序之后的序列b的第$\lfloor \frac{n}{2} \rfloor$ 位,序列的下标从0开始.

举例的话序列$\mathrm{middle}\{2,1\}$就是排序后$\{1,2\}$的第1位即2.

现在给长度为n的序列s,定义$s[l,r]$为s的从第l位到第r位组成的连续子序列. 现在有q个询问,每次询问$l \in [a,b], r \in [c,d]$的$\mathrm{middle}(s[l,r])$的最大值. 对所有询问,有$a < b < c < d$. 必须在线回答所有问题.

数据范围

5% $n, q \leq 100$

30% $n, q \leq 2000$

100% $n \leq 20000, q \leq 25000$

思考时间

算法

显然通过暴力算法是很难解决这个问题的,我们得分析一下中位数的性质.

我们不妨考虑二分答案,容易发现,一个长度为n的序列s,它的中位数$\geq x$的条件,就是$\geq x$的数的数量,大于等于$< x$的数的数量.

那么我们不妨将$\geq x$的数看成$+1$,$< x$的数看成-1,那么序列s的中位数$\geq x$的条件,就是转换之后该序列的和≥ 0.

进一步的,要判断$l \in [a, b], r \in [c, d]$的区间中,有没有中位数$\geq x$的,只需要判断转换之后,这些区间中有没有和$\geq 0$的.

这个只要求出$l \in [a, b], r \in [c, d]$的区间中和最大的,就可以判断了.

由于$a < b < c < d$,那么实际上这个区间可以看成$[l, b] + [b + 1, c - 1] + [c, r]$,也就是$[a, b]$的最大后缀和,$[b + 1, c - 1]$的和,$[c, d]$的最大前缀和.

算法

根据前面的分析,我们在判断答案是否可能$\geq x$的时候,只需要将所有数重新标号成$+1,-1$,然后求几个区间的和或最大前缀或最大后缀和即可. 由于可能的答案只有$O(n)$个,我们对数列中的每个值,都重新标号之后使用线段树来维护.就能在$O(\log n)$的时间内判断答案是否可能$\geq x$,那么因为同时需要二分$O(\log n)$次,故每次询问复杂度为$O(\log^2 n)$.

算法

不妨将s中的数进行排序,那么关于s_i的线段树,和关于s_{i+1}的线段树,只有一个位置被修改了.

那么套用我们之前提到的可持久化线段树,只需要$O(n \log n)$的时间和空间,就能得到关于所有s_i的线段树.

那么总时间复杂度就是$O(n \log n + q \log^2 n)$.

可以圆满的解决问题.

例题:kth xor

给$n(n \leq 10^5)$个数$[a_0, a_1, ..., a_{n-1}](a_i \leq 10^9)$,有$q(q \leq 10^5)$个询问.

每次询问$[a_l, a_{l+1}, ..., a_r]$这些数,全部xor上一个数x之后,其中的第k大的数是多少.

x, k对于每个询问都不同.

思考时间

例题:kth xor

给你一个Trie,仿照线段树,我们可以快速求出xor上x之后,第k大的数是多少.

从根往下递归即可.

那么我们令at_i为加入$[a_0, a_1, ..., a_{i-1}]$这些之后的函数式Trie.

那么询问$[a_l, a_{l+1}, ..., a_r]$时,只需要对$at_{r+1} - at_l$进行操作即可.

复杂度$O(n) - O(1)$.我们认为数的位数是常数.

可持久化平衡树

可持久化平衡树,顾名思义就是维护历史版本的平衡树.

我选择了较为容易实现的Treap来讲解.

Treap

Treap是一种平衡树,它的特性是每个点有一个随机权值,同时父亲的权值一定比孩子的权值小.

那么这棵平衡树,可以看成是按这随机权值顺序依次插入的普通平衡树.

由于插入顺序是随机的,故树高是期望$O(\log n)$的.

一些记号

若t表示某节点.

那么$\text{left}(t), \text{right}(t)$分别表示左右孩子.

$\text{size}(t)$表示以t为根的子树大小.

$\text{key}(t)$表示t的随机权值.

可持久化

为了方便讨论,我们将所有Treap的操作都简化为两个操作的组合:merge(a, b), split(a, n)

merge(a, b)考虑两个Treap:a, b,a中所有元素都比b中的小,要返回一个Treap,里面包含了a, b的所有元素.

split(a, n)考虑一个Treap:a,返回两个Treap,$left$, $right$,分别包含a的前n个元素和之后的其它元素.

那么插入可以看成先按给定位置split成两个部分,然后在中间放一个单元素Treap,然后依次merge.

删除可以看成按给定位置和之前之后split成三个部分,然后将第一部分和第三部分merge.

其他的平衡树共有操作保持原样.

merge的实现

考虑如何实现merge(a, b)

首先考虑a和b有一个为空树的情况,那么返回不为空树的那个即可.

其次若key$(a) <$ key(b),那么让a的左孩子不变,右孩子改为merge(right$(a), b$).

否则b的右孩子不变,左孩子改为merge$(a,$ left$(b))$.

注意到由于是可持久化的实现,修改是通过新建一个节点来实现的.

split的实现

考虑如何实现split(a, n)

如果$cnt = \text{size}(\text{left}(a)) \leq n$的,那么令$\{l, r\} = \text{split}(\text{left}(a), n)$

那么我们将a的左孩子改为r,返回$\{l, a\}$即可.

否则的话,令$\{l, r\} = \text{split}(\text{right}(a), n - cnt - 1)$

我们将a的右孩子改为l,返回$\{a, r\}$即可.

注意到由于是可持久化的实现,修改是通过新建一个节点来实现的.

一些操作的实现

那么提取一段,就可以通过2次split来得到.

删除一段,可以通过2次split和一次merge得到.

当然也有常数更小的写法,不过这么写比较方便.

超级编辑器

题目大意:

给一个字符串,你需要支持一些操作

插入一段字符串

复制内部的一段字符串,并插入到某给定位置.

询问一个位置的字符.

输入总大小不超过1MB.

字符串总大小不超过512MB,但是内存限制只有100MB.

超级编辑器

我们发现使用可持久化Treap可以实现这些操作并且内存只跟操作次数有关.

但是注意到一个很严重的问题,比如说我不停复制一个串,那么大家的随机权值都一样,Treap就彻底失去平衡性了.

怎么办呢,注意到只有在merge中才有使用随机权值,反正随机权值不过是个概率问题,我们不妨直接使用随机概率!

考虑merge(a, b),注意到a的随机权值是size(a)个数中最小的,b的随机权值是size(b)个数中最小的,那么key$(a) <$ key(b)的概率就是$\frac{\text{size}(a)}{\text{size}(a)+\text{size}(b)}$, 直接按这个概率进行合并.

能够通过各种各样的数据,但是本人能力有限,并没有给出详细证明其期望复杂度的能力,如果有人能给出详细证明或构造出使得该方法运行缓慢的数据请联系我.

凸包

下面介绍一些凸包的基础知识.

给定平面上n个点$p_0, p_1, ..., p_{n-1}$,这些点的凸包定义为一个包含这些点在内部的,面积最小的凸多边形.

上凸壳:这些点和$(0, -\infty)$组成的凸包.

下凸壳:这些点和$(0, \infty)$组成的凸包.

完全动态凸包

动态凸包问题是,给定一个点集合,要求支持插入,删除一个点,同时维护整个点集合的凸包.

只有插入

只有插入的情况是一个经典问题,我们可以用两个平衡树分别维护上凸壳和下凸壳.

删除

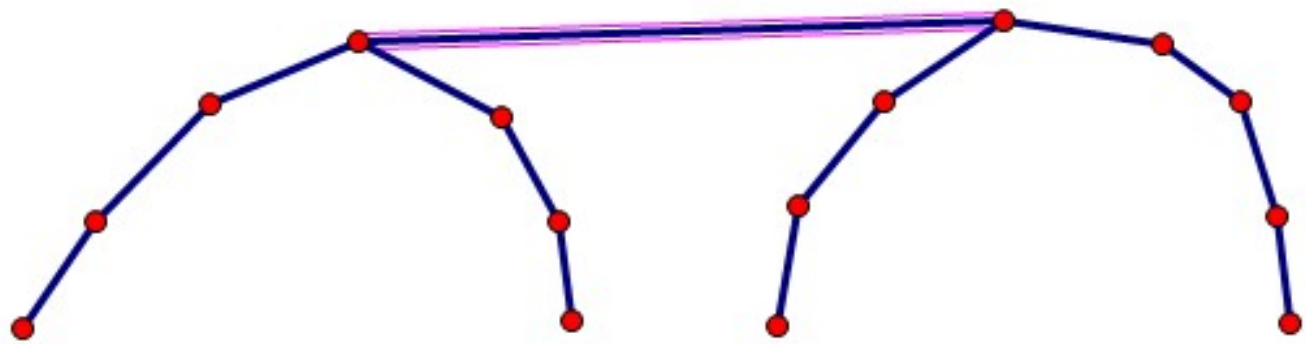
有删除的情况难度就很大了,因为一次删除可能会导致凸包上$O(n)$个点发生改变,暴力算法显然不能成功.

不妨继续考虑分别维护上凸壳下凸壳.

由于上凸壳下凸壳等价,我们只考虑上凸壳.

合并两个不相交的上凸壳.

考虑两个在x轴上不相交的上凸壳,将它们合并.



注意到我们只需要将这两个凸壳的外公切线求出之后,各自取一段放在一起即可.

那么我们只需要用可持久化平衡树来维护这两个上凸壳,不记求公切线复杂度的话,就能做到$O(\log n)$合并2个上凸壳.

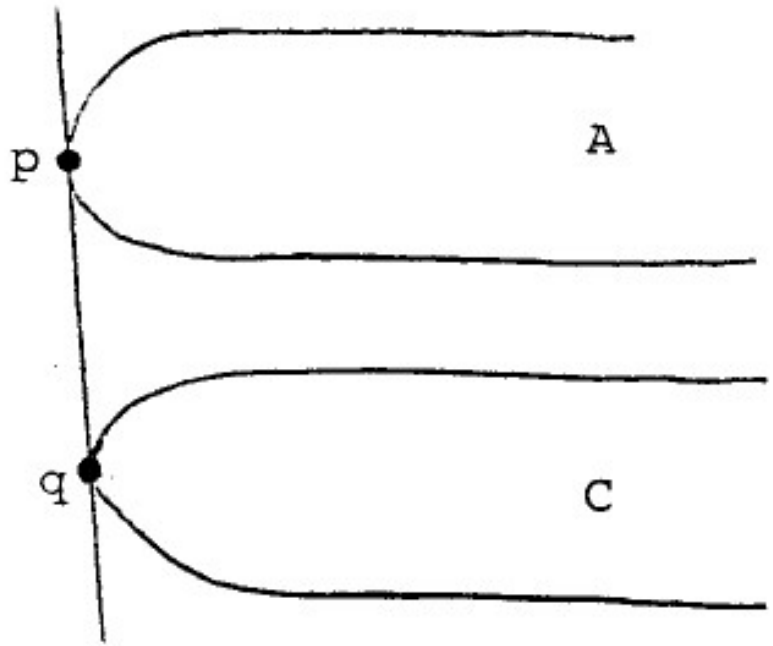
求两个凸壳间的外公切线

不妨设p, q分别为凸壳A, C上的一个点,我们来分情况进行各种讨论.

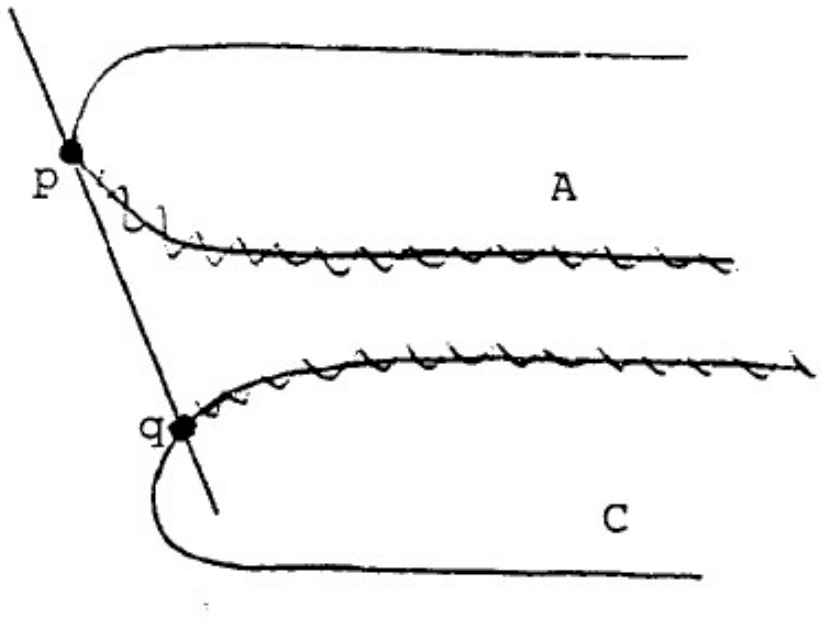
设公切线交凸壳A, C分别于u, v.

由于竖着影响排版,不妨把图片横过来.

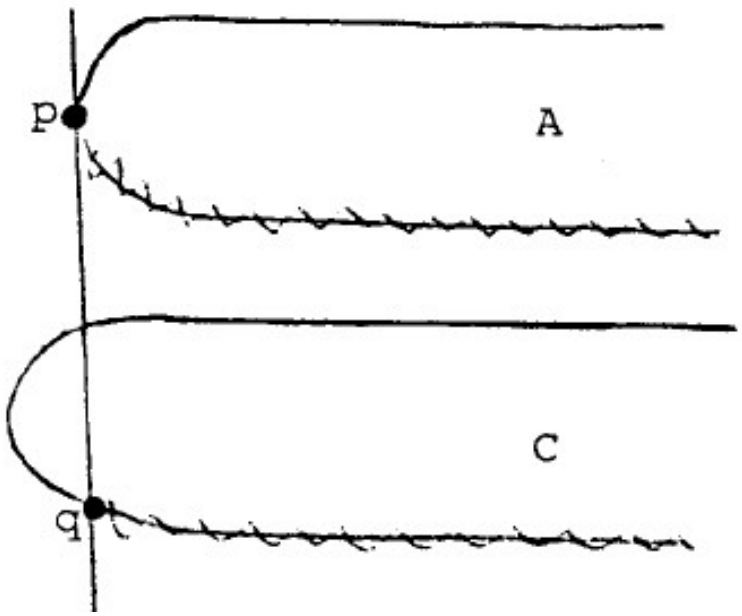
case 1:



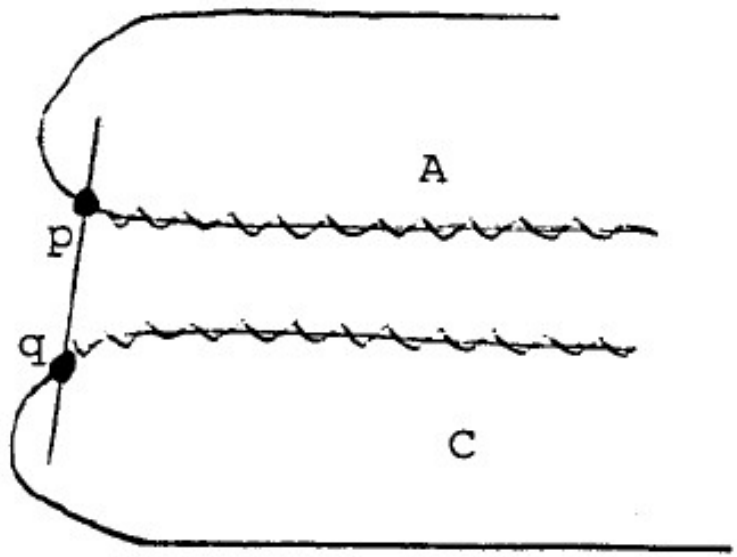
case 2:



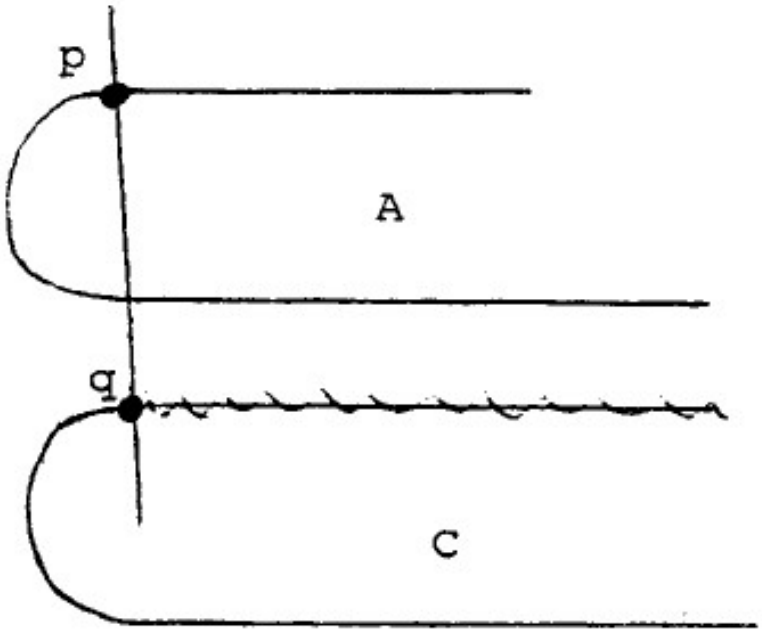
case 3:



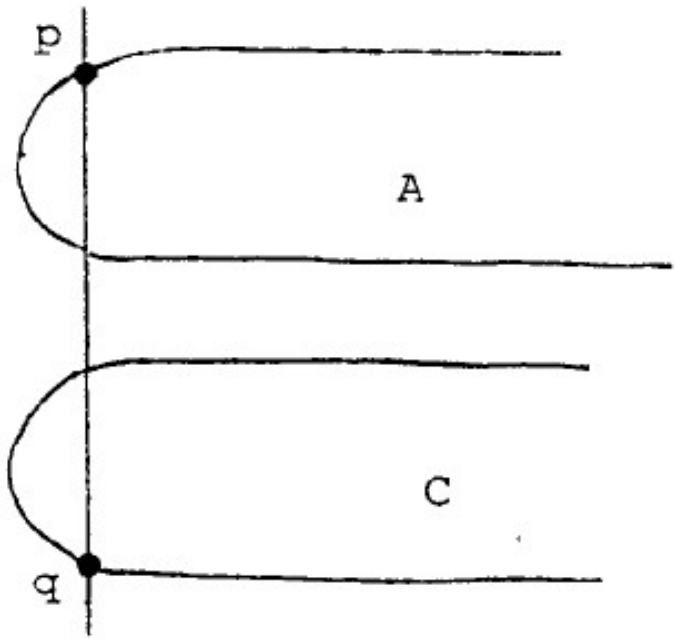
case 4:



case 5:



case 6:



求两个凸壳间的外公切线

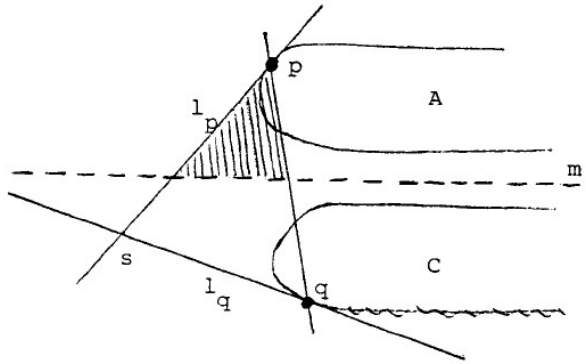
可以看出case 1我们直接就得到结果,case 2 $\sim$ 5,都直接可以判断出一些凸壳的部分(波浪)可以被舍弃,具体分析比较繁琐就略去不谈,大家可以自己思考.

但是在case 6中,我们无法直接确定该删去哪些部分,因为比方说u在A的最凸处,v即可能在q的左边,也可能在q 的右边.

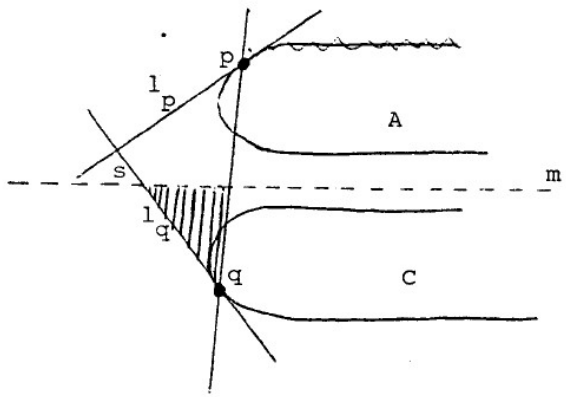
那么我们继续分两种情况讨论.

我们做一条p关于A的切线lp,q关于C的切线lq,设他们的交点为s.同时令一条凸包间的划分线为m.

case 6.1:s在m的下方.



case 6.2:s在m的上方.



容易发现图中的波浪部分都可以被删去.

求两个凸壳间的外公切线

那么如果给出两个凸壳的平衡树表示,不妨令p为一个的根,q为另一个的根.

可以看到每次操作都能使得p或q走到它的一个孩子,那么总时间复杂度就是$O(\log n)$.

实现删除操作

可以发现我们已经可以在$O(\log n)$的复杂度内合并两个上凸壳了.

那么我们用一个平衡树维护所有点按x轴的顺序,每个节点维护上凸壳和下凸壳.

插入删除只要在平衡树内做即可.树根的上凸壳和下凸壳就是最终的结果.

复杂度是平衡树的复杂度乘上合并的复杂度,故是$O(\log^2 n)$.

实现的一些细节问题

首先需要注意的是,在判断的过程中,我们需要知道p, q相邻的凸包上的点.

这个需要在$O(1)$时间内知道,我们不妨将其记录在点上.

那么注意到合并的时候.连线的那两点的左右相邻点信息改变了,所以这两个点要新建出来.

总结

这个东西代码难度和代码量都大出*了,在OI中有实际价值吗?

我们获得了理性的愉悦,视野的开阔.

一旦考场上遇到了并且你会,你就能"帅气的做出一道很难的计算几何题".

有理有据,令人信服!

例题:Almost

题目大意

一个长度为$n(n>1)$的数列$a_0, a_1, ..., a_{n-1}$的几乎平均数,定义为$\frac{\sum_{i=0}^{n-1} a_i}{n-1}$.

q个询问:$a_l, ..., a_r$的连续子序列中,最大的几乎平均数是多少.

数据范围

$n \leq 10^5, q \leq 3 * 10^4$.

例子

算法

这题是艾雨青神犇集训队互测比赛的题目,当时他给出的算法每次询问是$O(\sqrt{n}\log n)$的.

转化成几何问题

我们令$sum_i = \sum_{k=0}^{k=i} a_i$.

同时令$R_i = (i, sum_i), L_i = (i, sum_{i-1})$.

那么可以发现区间$[l, r]$的几乎平均数,就是$L_l \to R_r$的斜率.

我们的目标就是求$l \le i < j \le r$的$L_i \to R_j$的最大斜率.

L_i, R_i的x坐标是递增的.

算法1

现在我们给出一个每次询问$O(\log^2 n)$的算法.

注意到如果我们要求$i \in [a,b], j \in [c,d], b < c$的$L_l \to R_r$的最大斜率.

其实就是$[a,b]$组成的L_*的下凸壳,跟$[c,d]$组成的R_*的上凸壳之间的公切线的斜率,对于上下凸壳间的公切线,我们仿照之前的算法,也可以给出一个$O(\log n)$的算法.

那么就能在$O(\log n)$的时间内得出上述问题的结果.

那么我们维护一个线段树,对于一个线段树范围是$[l,r)$的节点,

我们维护$l \le i < j < r$的$L_i \to R_j$的最大斜率和R_*的上凸壳以及L_*的下凸壳.

算法1

前者可以通过孩子的答案和一次凸壳间的公切线得到.后两者直接通过孩子合并出来即可.

那么建树的复杂度就是$O(n \log n)$.

考虑询问$[l, r]$,不妨将其按顺序拆成$k = O(\log n)$个线段树节点$t_0, t_1, \ldots, t_{k-1}$.

那么答案区间有两种可能,一种是两个节点间,一个是一个节点内部,后者我们可以直接得出.

前者的话考虑答案区间右端点在t_i内部,我们在此时维护$t_0, t_1, \ldots, t_{i-1}$的$L$的下凸壳的并.

那么$O(\log n)$时间就能得出此情况下的最优结果.

同时再将t_i也合并进去即可.

复杂度是$O(n \log n) + O(\log^2 n)$.

算法2

接下来我们再给出一个$O(\log n)$的算法

我们将所有数分成$O(\sqrt{n})$块,每块大小$O(\sqrt{n})$.

我们预处理出每块的前i个组成的R_*的上凸壳,后i个组成的L_*的下凸壳,以及前缀和后缀分别的内部答案. 每块的复杂度是$O(\sqrt{n}\log n)$,那么这部分复杂度就是$O(n\log n)$.

同时我们再预处理出第i块到第j块的L_*下凸壳,R_*上凸壳,和内部的区间的答案.

第i块到第j块的结果可以通过第i块到第$j-1$块的结果在$O(\log n)$的时间内得出.

那么这部分的复杂度就是$O(\sqrt{n^2}\log n) = O(n\log n)$.

算法2

考虑询问$[l, r]$,如果它不是在某块的内部,那么它被分成了三部分:某块的后几个A,中间连续几个块B,某块的前几个C.

那么答案就可能是在部分内部(已处理),AB之间,AC之间,BC之间,分别计算即可.

如果它在某块的内部,我们对每个块递归使用以上算法进行预处理.

考虑若对于大小n的,预处理需要的时间是$F(n)$

那么$F(n) = \sqrt{n}F(\sqrt{n}) + n\log n$.

不妨设$F(n) = kn\log n$.

算法2

那么

$$F(n) = \sqrt{n}k\sqrt{n}\log\sqrt{n} + n\log n$$

$$\log\sqrt{n} = \frac{\log n}{2}$$

$$kn\log n = \frac{k}{2}n\log n + n\log n$$

$$k = 2$$

也就是说$F(n) = O(n\log n)$.

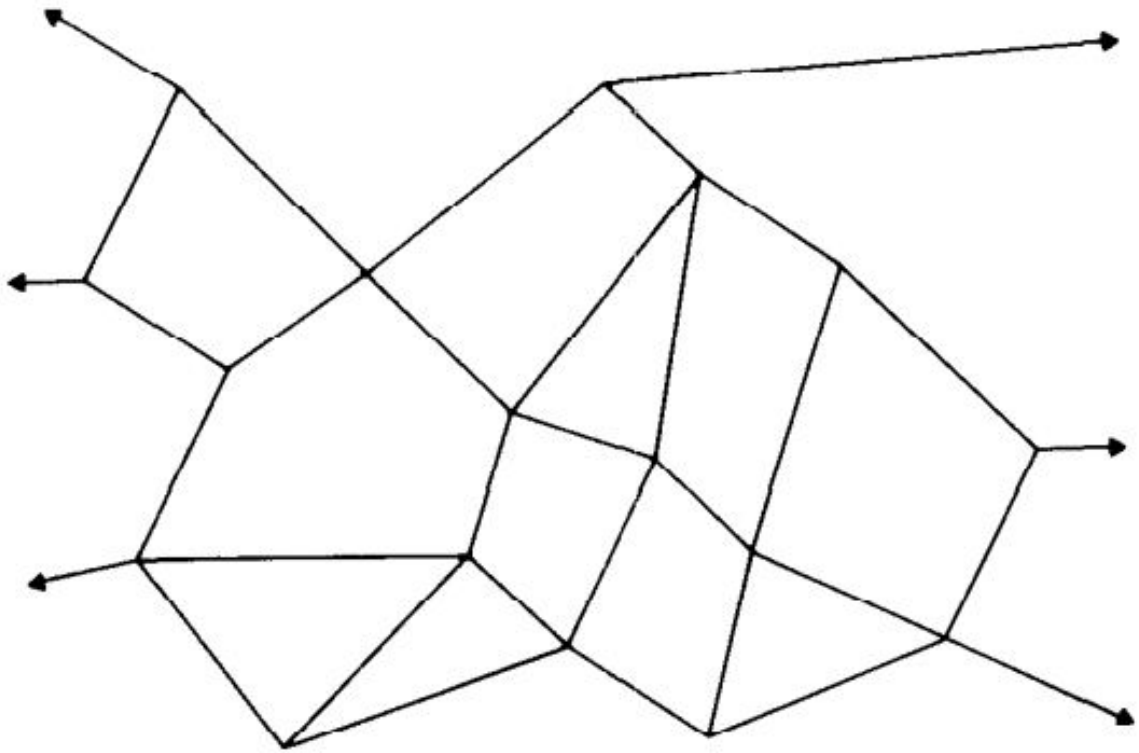
总复杂度$O(n\log n) + O(\log n)$.

当然到大小递归到非常小的时候(比如小于16),不需要分块,暴力即可,复杂度为常数,不影响总体复杂度.

点定位问题

给出n个只能在端点处相交的线段,它们将平面划分成了很多个区域.

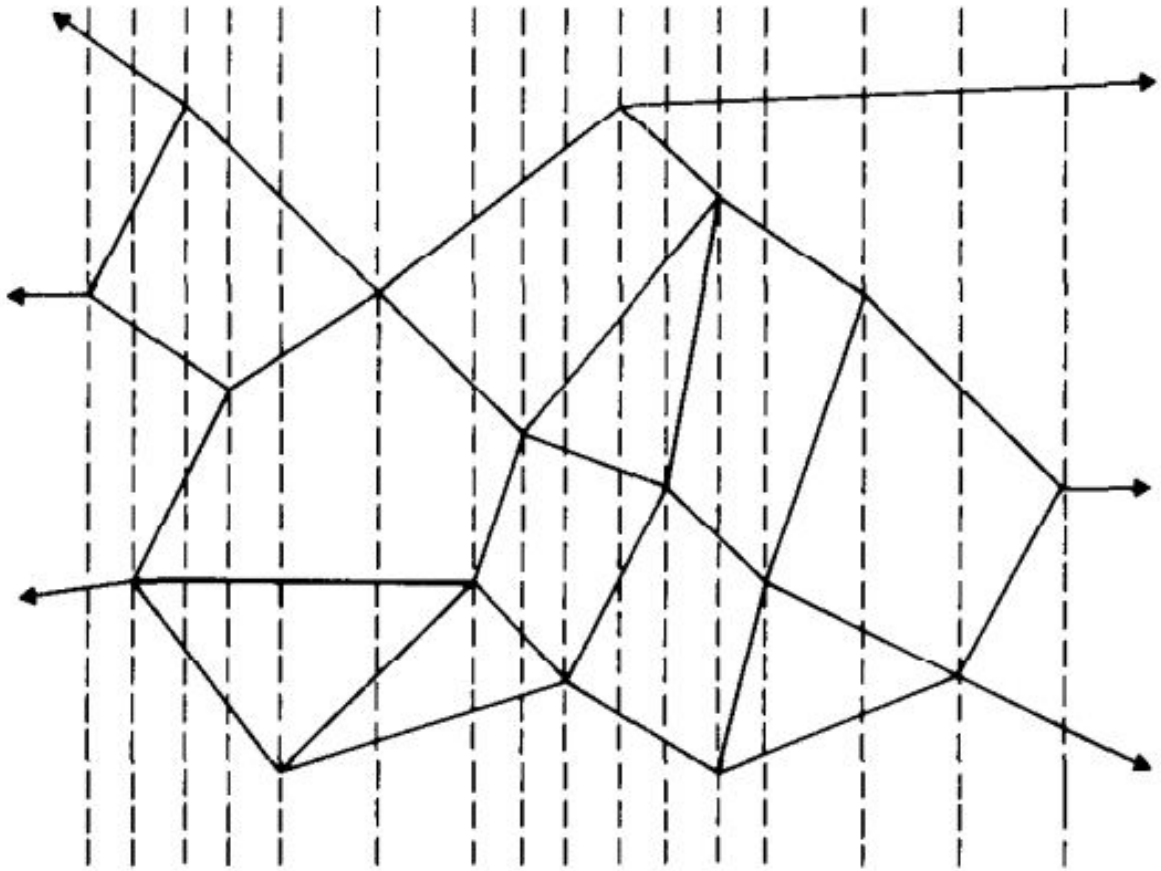
每次询问一个给定点属于哪个区域.要求在线.



点定位问题

由于线段只能在端点处相交,不妨假定没有跟y轴的线段,我们可以看到任何两个线段上下关系是不变的.

上下关系只会在一个线段被插入或删除时发生变化.



点定位问题

我们使用可持久化平衡树维护每个区间内部的线段的顺序,下个区间的顺序可以在该区间添加或删除几个线段得到.

那么通过$O(n \log n)$的预处理,就能得出每个区间的线段的顺序的平衡树.

询问时我们确定点所在的区间,然后找出所在的位置即可.

例题1

给$n*n(n\leq 100)$的矩阵,每次询问一个子矩形中第k大的数.

时限3s,询问次数≤ 5000000,必须在线.

思考时间

例题1

预处理$a[i][j]$表示以(i,j)为右上角,$(0,0)$为左下角的矩阵内部所有数对应的权值线段树.

$a[i][j]$可以通过$a[i][j-1]$或者$a[i-1][j]$,在$O(n\log n)$的时间内算出.

预处理复杂度就是$O(n^3\log n)$.

一个子矩形可以表示成4个$a[*][*]$的代数和.

询问复杂度$O(\log n)$.

例题2

给一棵树,支持两个操作,一个是修改某条边的权值,一个是询问第i次修改后某两点之间的最大边权.

必须在线.

思考时间

例题2

使用树链剖分,并且对每个树链剖分中的线段树,维护它的历史版本即可.

例题3

给一个字符串,支持三个操作,一个是后面添加一个字符,一个是回到第i次操作之后,一个是询问某个串的出现次数.

时限3s,字符串长度不会超过20w,操作数不超过10w.

思考时间

例题3

后缀自动机和后缀树很难实现可持久化,我们不妨使用后缀数组.

把所有字符串反过来,那么后面添加字符变成了往前面添加字符.

$S \to cS$

可以看到后缀数组中插入了一个新后缀cS.

我们使用平衡树维护后缀数组,使用Hash就可以在$O(\log n)$的时间内比较两个串的大小,那么插入一个串的复杂度就是$O(\log^2 n)$.

同时平衡树是可以可持久化的,所以就完成了可持久化的拓展.

例题4

维护两个字符串集合S,T，一开始S和T都只有一个空串,编号都为1,要求支持操作:

1.在S的某一个串S_i后添加一个字符c，加入S

2.在T的某一个串T_i的前面或后面添加一个字符c，加入T

3.将T的两个串T_i,T_j首尾相接形成一个新串T_iT_j，加入T

4.询问T中的某个串T_i在S中某个串S_i中的出现次数.(如果T_i是空串，输出0)

询问≤ 300000,c是小写拉丁字母.

操作1 ≤ 100000

操作3 ≤ 30000

操作4 ≤ 100000

不要求在线.

思考时间

离线解决问题.

可以看到所有S形成了一个Trie. 建出这个Trie的后缀自动机SAM.

对每个T_i,维护它对应的SAM中的节点和匹配长度.

对于操作2,往前和往后加一个字符,是SAM的经典操作,$O(1)$.

对于操作4,我们需要知道T_i对应的节点中的Right集合中,有几个是S_j在Trie中的祖先,由于Right集合形成一棵树,那么求出Right集合的树的dfs序,当前这个Right集合就对应了一个区间,我们要找出这个区间内部有哪些节点是S_j的祖先,可以使用可持久化线段树解决,$O(\log n)$.

对于操作3,我们注意到,SAM中的每个节点,对应了Trie的逆序(正序是从根到叶子,逆序是从叶子到根)的后缀数组的一个区间.

我们用可持久化平衡树维护T_i,那么在Trie的逆序的后缀数组中二分查找出合并后的T_i T_j对应的区间,从而得到对应的节点即可$O(\log^2 n)$.

例题4:在线

在刚刚的题目中,我们离线使用后缀自动机得出了一个优秀的算法.

那么这个题目一定要求在线的话,又该怎么做呢?

例题4:在线

考虑S_i构成的Trie,令Up_u表示节点u到根的路径上的字符依次排列形成的字符串.

那么询问T_i在S_i中出现了几次,若S_i在Trie中对应u点.

那么就是询问u在Trie中的所有祖先中(包括本身),有几个的Up开头是T_i.

我们对每个u,用可持久化平衡树Set_u维护它所有祖先的Up的有序集合.

那么以某个特定前缀T_i开头的必然是一个区间,我们二分就能获得答案.

Set_u可以通过u的父亲的Set加入Up_u得到.

故操作1,4复杂度为$O(\log^2 n)$.

2,3复杂度为$O(\log n)$

例题4:在线

我们在需要比较两个串的大小,不妨使用Hash.

但是注意到,为了能够$O(\log n)$求出T_i跟Up_u的lcp长度来比较他们的大小,我们需要能够在$O(1)$的时间内得到Up_u的某一个前缀的hash值.

不妨令我们需要知道Up_u的长度为len的前缀的hash值,那么我们只要预处理跟到点的hash值,然后得到u的第len个祖先即可.

我们需要在$O(1)$的时间内回答,某个点的第len个祖先这个问题.

我们预处理一个点往上第$1 \sim \sqrt{n}$个祖先和第$k \cdot \sqrt{n}(k \leq \sqrt{n})$个祖先.就能在$O(1)$得出结果.

题外话

在线回答一个点的某特定深度的祖先,这个问题是一个经典问题.

有$O(n) - O(1)$的做法.

非常NB,大家可以自己搜论文看看.

题外话

如果一个可持久化线段树,每次修改一个节点,并要求访问历史版本,有总空间$O(n)$的做法.

结束语

欢迎继续提问.有更多问题的可以找我课后交流.

谢谢大家.