数据结构选讲

董炜隽

广州市第六中学

2018 年 3 月 29 日

线段树

相信大家都会。

HEOI2013 Segment

要求在平面直角坐标系下维护两种操作：

- 在平面上加入一条线段。
- 给一个数 k，询问与直线 $x = k$ 相交的线段中，交点最靠上的线段的编号。

$n \leq 10^5$。

HEOI2013 Segment

考虑在线段树的一个区间上维护一条线段作为标记。在一个结点加入一条线段时，如果结点上没有标记，那么直接把这条线段作为标记。结点上已有标记时，先判一下两条线段是否有一条完全在另一条上方，如果没有的话就把在中点处较高的线段作为该结点的标记，另一条线段再下传到对应的儿子。查询时把路径上所有标记取个 max 即可。

修改复杂度 $O(\log^2 n)$，查询复杂度 $O(\log n)$。

清华集训 2012 楼房重建

工地上有 n 栋楼房，第 i 栋楼房可以用一条连接 $(i,0)$ 和 (i,h_i) 的线段表示。有 m 天，第 i 天建筑队会把横坐标为 x_i 的房屋的高度修改为 y_i。问每天建筑队完工后，站在 $(0,0)$ 处能看到多少栋楼房。

$n,m \leq 10^5$。

清华集训 2012 楼房重建

考虑对于每栋楼算出 $\frac{h_i}{i}$，问题转化求关于斜率的严格上升序列的长度。

我们对线段树的每个区间，维护该区间内的上升序列长度 len 以及最大值 $maxw$。合并信息时需要求出右子区间内起点大于左子区间最大值的上升序列的长度。考虑这个问题要怎么解决，对于一个区间，如果查询值大于等于左子区间最大值，那么直接递归右子区间；否则递归左子区间，再把长度加上该区间的 len 减左子区间的 len。

时间复杂度 $O(n \log^2 n)$。

重量平衡树

- treap

  插入/删除时，期望旋转次数 $O(1)$，旋转的子树大小期望 $O(\log n)$。

重量平衡树

- treap

  插入/删除时，期望旋转次数 $O(1)$，旋转的子树大小期望 $O(\log n)$。

- 替罪羊树

  不需要旋转。

应用

- $O(1)$ 查询序列相对顺序

应用

- $O(1)$ 查询序列相对顺序
- 维护后缀平衡树

应用

- $O(1)$ 查询序列相对顺序
- 维护后缀平衡树
- 在内层套数据结构

应用

- $O(1)$ 查询序列相对顺序
- 维护后缀平衡树
- 在内层套数据结构
- 用替罪羊树的思想重构动态点分治

誓约

这是一道交互题。

给一种满足交换律和结合律的运算 $F(x,y)$，可以通过调用交互库的函数来进行这种运算。

要求维护一个序列，支持：在末尾插入/删除一个元素，询问区间元素做 F 运算的结果。每次询问最多只能调用一次 F。另外还需要支持可持久化。

操作数不超过 $3*10^5$，空间限制 2G。

誓约

考虑维护一个 treap，每个结点维护它往左/右子树的后缀/前缀 F。查询时找到把 l, r 划分到两边的那个节点就可以了。

插入的时候需要把祖先的右子树的前缀 F 都加上一个元素，这一部分直接可持久化的话会导致复杂度再多一个 log。注意到所有插入都在末尾，因此查询时我们可以追溯到一个祖先版本，使得查询的是一个序列的后缀。那么对于所有的右儿子我们只需要记录整个区间的 F 即可。

时空复杂度都是 $O(n \log n)$。

finger search

finger search

- splay

  在 n 个点的 splay 上执行 m 次插入/删除/访问操作的均摊复杂度是 $O(n+\sum_{i=1}^{m}\log d_i)$，其中 d_i 为每次操作的元素与上一次操作的元素的排名之差。

finger search

- splay

  在 n 个点的 splay 上执行 m 次插入/删除/访问操作的均摊复杂度是 $O(n+\sum_{i=1}^{m}\log d_i)$，其中 d_i 为每次操作的元素与上一次操作的元素的排名之差。

- treap

  两个排名相差 d 的元素在 treap 上的路径长度是期望 $O(\log d)$。

启发式合并

通常的平衡树启发式合并是两个 log 的。

假设我们要合并两个大小分别为 $n, m(n \geq m)$ 的平衡树，可以用 finger search 把较小的那棵平衡树按升/降序插入另一棵平衡树中。由于 log 是凸的，因此复杂度是 $O(m \log \frac{n}{m})$。那么启发式合并的复杂度就能做到一个 log。

树上长度小于 k 路径计数

给一棵 n 个节点的树，边有边权，问有多少对点的距离小于 k。

树上长度小于 k 路径计数

给一棵 n 个节点的树，边有边权，问有多少对点的距离小于 k。

经典做法是点分治，复杂度是 $O(n\log^2 n)$。

树上长度小于 k 路径计数

给一棵 n 个节点的树，边有边权，问有多少对点的距离小于 k。

经典做法是点分治，复杂度是 $O(n\log^2 n)$。

考虑 dfs，对每个结点维护一棵平衡树表示它到子树内每个结点的距离。相当于要在每次合并两棵平衡树之前计算它们之间有多少个元素对的和小于 k，这个同样可以用 finger search 完成。时间复杂度为 $O(n\log n)$。

清华集训 2015 小 Q 与找茬

这是一道交互题。

初始时给你 n 个二维平面上的点。之后会有 m 次询问，每次询问会给出一个矩形，你需要尽可能多地找出在这个矩形内部的点。最终会按照你在时间限制内找出的总点数来评分。

$n \leq 5 * 10^5, m \leq 2 * 10^6$，时间限制 3s。

清华集训 2015 小 Q 与找茬

这是一道交互题。

初始时给你 n 个二维平面上的点。之后会有 m 次询问，每次询问会给出一个矩形，你需要尽可能多地找出在这个矩形内部的点。最终会按照你在时间限制内找出的总点数来评分。

$n \leq 5 * 10^5, m \leq 2 * 10^6$，时间限制 3s。

使用划分树就可以在 $O(\log n + k)$ 的复杂度内找出查询矩形中的 k 个点。

划分树

时代的眼泪

划分树

时代的眼泪

对线段树的每个区间维护一个有序表，另外再维护每个元素在左/右子区间的有序表内的前驱。查询时只需要在根结点的有序表上二分一次就可以了。

BZOJ NOI 十连测 2016 幻想

要求维护一个字符串，支持在末尾增删一个字符，以及查询给定的串在某个区间中出现了多少次。强制在线。

每个时刻字符串的长度不超过 $2 * 10^5$，操作数不超过 $5 * 10^5$，查询串总串长不超过 $5 * 10^6$。

BZOJ NOI 十连测 2016 幻想

由于需要同时在一端增删字符，可以想到用后缀平衡树来维护。对于后缀平衡树上的每个结点我们用一个 vector 存下子树中所有后缀的位置，询问的时候二分一下就好了。由于后缀平衡树本身是用重量平衡树维护的，因此在树的形态改变时可以用暴力归并出新的 vector。

这样做插入的复杂度是 $O(\log^2 n)$，查询的复杂度是 $O(|S| \log n + \log^2 n)$。考虑进一步地优化每个部分的复杂度。

BZOJ NOI 十连测 2016 幻想

替罪羊树在重构时需要归并出整个子树的 vector，这样会多出一个 log。我们可以改为使用 treap，这样只需要重构旋转的结点上的 vector，插入就可以降到 $O(\log n)$。

查询的时候在每个结点上都需要用二分哈希来求与询问串的 LCP。我们可以通过在后缀平衡树上额外记录一些 LCP 的信息，来把 $|S|$ 上的这个 log 去掉。

最后我们再用划分树的思想，记录 vector 里的每个元素在儿子的 vector 内的前驱，这样就只需要在根节点上二分一次了。

最终插入的复杂度是 $O(\log n)$，查询的复杂度是 $O(|S| + \log n)$。

k-d tree

一维的 k-d tree 其实就是线段树。

对于高维的情况，可以循环地以每维坐标作为划分依据，把中位数所在的点作为该子树的根。查找中位数可以直接调用 STL 的 nth_element()。

建树的复杂度是 $O(n \log n)$，并且树的深度是 $O(\log n)$ 的。

插入与删除

k-d tree 在插入时会导致树不再平衡。可以使用重量平衡树的方法来重构子树。插入的复杂度是 $O(\log^2 n)$。

删除的话可以直接延迟删除。

范围操作

k-d tree 可以较快速地对一个平行于坐标轴的空间范围内的点进行操作。

从根开始遍历，若当前结点的子树表示的空间完全在操作范围内，则直接对该结点进行操作，若与操作范围没有交，则直接回溯，否则继续递归两个儿子。

范围操作

k-d tree 可以较快速地对一个平行于坐标轴的空间范围内的点进行操作。

从根开始遍历，若当前结点的子树表示的空间完全在操作范围内，则直接对该结点进行操作，若与操作范围没有交，则直接回溯，否则继续递归两个儿子。

复杂度如何计算？考虑分开算每一维的贡献：

$$T(n) = O(2^k) + 2^{k-1} T\left(\frac{n}{2^k}\right) = O(n^{1-\frac{1}{k}})$$

因此时间复杂度就是 $O(kn^{1-\frac{1}{k}})$。

查询最近点

查询最近点时可以用 k-d tree 来进行剪枝，但是复杂度是可以被卡到 $O(n)$ 的，例如画一个圆然后查询圆心。

不过当 n 远大于 2^k 时对随机数据询问的复杂度是期望 $O(\log n)$ 的。

PA2011 Kangaroos

给出一个长度为 n 的区间的序列。有 m 次询问，每次询问序列中最长的连续子序列，使得子序列中的每个区间都与 $[l, r]$ 有交。

$n \leq 5 * 10^4, m \leq 2 * 10^5$。

PA2011 Kangaroos

考虑把每次询问视为二维平面上的一个点 (l_i, r_i)，那么与某个区间相交的询问都位于一个矩形内。我们依次枚举序列上的区间，每次把与这个区间相交的询问的权值加一、不与这个区间相交的询问的权值置为 0。那么最后每个询问点的历史最大值就是答案了。

时间复杂度 $O(m \log n + n\sqrt{m})$。

集训队互测 2014 War

有 k 个阵营，每个阵营有若干个炮塔。第 i 个炮塔可以攻击到离它欧几里得距离小于等于 r_i 或者曼哈顿距离小于等于 a_i 的炮塔。共有 m 轮攻击，每轮会随机选一个炮塔来攻击它范围内的所有不同阵营的炮塔。问 m 轮后期望剩下多少个从未被攻击过的阵营。注意炮塔被攻击过后仍然能攻击别人。

$n, m, k \leq 35000$。保证在数据生成时不考虑一个点的具体位置，而将它的横纵坐标分开考虑。

集训队互测 2014 War

实际上只需要求出每个阵营能被多少个炮塔打到就能算出答案了。

先考虑炮塔的阵营互不相同时怎么做。可以用 k-d tree 来把每个炮塔攻击范围内的点的答案 +1。由于题目里的那个限制所以可能不会被卡掉。

当炮塔的阵营可能相同时，可以用 bitset 在 k-d tree 上打永久化标记，做完所有修改后再把标记下推一遍，然后把同个阵营里所有点的 bitset 求个并就可以了。

树链剖分与 LCT

树链剖分与 LCT

- 链操作
- 子树操作
- 提供一个链分治的结构
  （例如动态维护带权重心）

树链剖分

树链剖分得到的序列本身就是一个优先遍历重儿子的 dfs 序，因此可以原生态地支持子树操作。

树链剖分

树链剖分得到的序列本身就是一个优先遍历重儿子的 dfs 序，因此可以原生态地支持子树操作。

如果有换根怎么办？

初始时先随便选一个点作为根。进行子树操作时讨论一下该子树实际对应的区间即可。

可以做一下 BZOJ3083。

树链剖分

其实树链剖分也是可以支持一些动态操作的。

树链剖分

其实树链剖分也是可以支持一些动态操作的。

link 一个子树时，检查一下到根的路径上的轻边会不会变成重边即可。

树链剖分

其实树链剖分也是可以支持一些动态操作的。

link 一个子树时，检查一下到根的路径上的轻边会不会变成重边即可。

cut 一个子树时，到根的路径上最多只会有 $O(\log n)$ 条重边变成轻边。考虑把重儿子重新定义为 $2s(v) > s(u)$ 的儿子 v，这样可以对每个点维护出 $2s(x) - s(fa(x))$，每次检查最小值即可。这时还需要知道是否会产生新的重边，因此还需要维护子树大小最大的儿子。

树链剖分

大部分情况下树链剖分需要使用线段树等数据结构来维护每条重链，这样复杂度是两个 log 的。

树链剖分

大部分情况下树链剖分需要使用线段树等数据结构来维护每条重链，这样复杂度是两个 log 的。

Q：静态的树剖为啥会比动态的 LCT 的复杂度还要高？

树链剖分

大部分情况下树链剖分需要使用线段树等数据结构来维护每条重链，这样复杂度是两个 log 的。

Q：静态的树剖为啥会比动态的 LCT 的复杂度还要高？

实际上用 splay 来维护重链就是一个 log 了，复杂度分析和 LCT 是一样的。

树链剖分

大部分情况下树链剖分需要使用线段树等数据结构来维护每条重链，这样复杂度是两个 log 的。

Q：静态的树剖为啥会比动态的 LCT 的复杂度还要高？

实际上用 splay 来维护重链就是一个 log 了，复杂度分析和 LCT 是一样的。

splay 常数太大了，有没有更好的做法？

树链剖分

考虑从每个点对应的线段树结点向轻儿子所在重链的线段树的根连一条边，变成一棵“虚拟树”。那么查询时相当于相当于走了一条“虚拟树”上的路径，因此我们需要使得“虚拟树”尽可能地达到一个全局的平衡。

树链剖分

考虑从每个点对应的线段树结点向轻儿子所在重链的线段树的根连一条边，变成一棵“虚拟树”。那么查询时相当于相当于走了一条“虚拟树”上的路径，因此我们需要使得“虚拟树”尽可能地达到一个全局的平衡。

我们对每条重链分别构建它的树。设这条重链自顶向下依次为 $a_1, a_2, \ldots, a_m$，定义 $s(x)$ 为 $1+$ 结点 x 的所有轻儿子的子树大小之和，那么我们找到一个最小的 k 使得

$$\sum_{i=1}^{k} s(a_i) \geq \frac{1}{2} \sum_{i=1}^{m} s(a_i)$$

然后我们把 a_k 作为树的根，接着再递归两边继续建树。

树链剖分

可以发现无论是在这样的树上走到一个儿子，还是走一条轻边，都会使子树大小减半。因此“虚拟树”的深度是 $O(\log n)$ 的。建树的复杂度是 $O(n \log n)$。

LCT

LCT 怎么维护子树信息呢？

考虑对于每个点维护出它通过虚边连到的子树的信息和，然后在 splay 上再把实链和虚子树的信息合并一下就好了。

如果维护的信息是可减的，那么虚实边切换时可以 $O(1)$ 维护。如果信息不可减的话（例如 $\min, \max$），就需要用堆或者其他数据结构来维护，这样复杂度会多一个 $\log$。

LCT

这里介绍一种小技巧。当信息在边上时，我们可以通过添加辅助点的方式把每个点的度数降到 3。这样在虚实边切换时就可以暴力维护了。

边本质上是度数为 2 的点，因此加了辅助点后路径上经过的边仍然不变。但是如果信息在点上的话就不能这么做了。

BZOJ3779 重组病毒

给一棵 n 个结点的树，每个结点有一个颜色，初始时每个结点颜色互不相同，并且根为 1。定义每个结点的权值为该点到根的路径上不同颜色的个数。有 m 次操作：

- 把 x 到根结点的路径上的所有点染成一种新的颜色。
- 把 x 到根结点的路径上的所有点染成一种新的颜色，并把 x 设为根。
- 查询 x 的子树内所有点权值的平均值。

$n, m \leq 10^5$。

BZOJ3779 重组病毒

可以发现其实就是在维护一个 LCT，同种颜色的点就是一条实链。在虚实边切换的时候，需要把某个子树的权值加减一下。

直接在 LCT 上维护子树和可能比较麻烦。考虑到树实际上是静态的，可以初始时随便选一个点作为根求出 dfs 序，然后用一个线段树来维护权值。修改/查询子树时再用和前面树剖换根一样的做法讨论一下就好了。

时间复杂度 $O(n\log n+m\log^2 n)$。

BZOJ3159 决战

给一棵 n 个结点的树，每个结点有点权且初始为 0。有 m 次操作：

- 把一条链的点权加 w。
- 查询一条链的点权和/最大值/最小值。
- 把一条链的点权翻转。

$n, m \leq 50000$。

BZOJ3159 决战

考虑用 LCT 来维护。最棘手的是链翻转点权的操作，我们不能直接把 LCT 的 splay 翻转，那样子会改变了树的形态。可以把形态和权值分开来维护，我们对 LCT 的每条实链再另外用一棵 splay 来维护权值。每次要合并/分裂形态的 splay 时就对应地对权值的 splay 做同样的操作，翻转权值时只翻转权值的 splay 就可以了。时间复杂度为 $O(n + m \log n)$。

当然用树链剖分也是可以做的。可以对每条重链维护一棵 splay，链翻转点权时把对应区间的 splay 拉出来拼在一起，并在根结点打上翻转标记，最后再切成一段段放回原位就好了。

Thanks.