

# 树套树与分治

——清华大学 nzhtl1477

# 自我介绍

- 洛谷上的ID: noip
- 其他地方的ID: nzhtl1477, ODT
- 只会讲数据结构
- THUWC2020, CTT2020, CTS2021, NOI2020, CTT2021, CTS2022, THUSC2022, CTT2022出题人
- ECfinal2020, ECfinal2021, 沈阳2021区域赛, 昆明2022区域赛, 济南2022区域赛出题人

# 联系

- 动态的一维问题：维护一个 $n$ 个点序列，支持修改和查询，这里分为三种：
  - 1.单点修改，区间查询
  - 2.区间修改，单点查询
  - 3.区间修改，区间查询
- 动态的二维问题：
  - 第一种：维护一个 $n*n$ 的数组，支持修改和查询
  - 第二种：维护二维平面上 $n$ 个点，支持修改和查询

# 联系

- KDT主要解决二维平面上的矩形修改，矩形查询问题
- 树套树主要解决二维平面上的单点修改，矩形查询或者矩形修改单点查询问题
- “CDQ分治”功能上和树套树是等价的，所以这里讲题的时候主要讲树套树做法，因为我们关注的是数据结构题的思维方法而不是具体实现细节

# 偏序维护

- 即每次对满足多维的一个限制的所有数进行操作
- 多维的限制：每个点 $i$ 有 $a_i, b_i, c_i \dots$ 等不同的属性
- 每次对 $l_1 \leq a_i \leq r_1, l_2 \leq b_i \leq r_2 \dots$ 的 $i$ 进行一次查询操作，或插入一个单点

# 如何维护？

- 1.Range Tree
- 2.K-D Tree
- 3.Quad Tree , Octree
- 4.一些其他的空间划分树

# Range Tree

- 其实就是狭义上的树套树

# 树套树

- 能在 $O(\log n^d)$ 的复杂度内进行一次 $d$ 维偏序的范围查询
- 能在 $O(\log n^d)$ 的复杂度内进行一次 $d$ 维偏序的单点修改
- 空间为 $O(n \log n^{d-1})$ ，可以优化到 $O(n(\log n / \log \log n)^{d-1})$ ，不过这个应该没人会所以可以无视掉



# 树套树

- 具体来说什么树套什么树是有关系的

# 树套树

- 如果要维护 $d$ 维, 出于方便设每维的值大小是 $v$ 的一个偏序

# 高维树状数组

- 本质就是树状数组的嵌套
- 时间复杂度 $O(\log v^d)$ , 空间复杂度 $O(v^d)$
- 可以通过预先高维离散化来做到
- 时间复杂度 $O(\log v^d)$ , 空间复杂度 $O(n \log v^d)$
- 不过这个基本上无意义

~

- 接下来只讨论二维情况
- 因为三维情况下常数和空间都起飞了， 就没见过一个三维的题

# 关于树套树

- 我最开始也觉得这东西很牛逼，很码农
- （后面发现也就5分钟的事。。。）
- 感觉能用到的树套树基本上都可以用树状数组套平衡树/线段树来写

# 函数化数据结构的思想

- 我们以树状数组套平衡树作为例子来介绍一下树套树

# 普通的树状数组

- 维护一个序列支持:
- 1.把x位置的值加上y
- 2.查询一个区间的和

```
inline void modify( int x , int y )
{
    for( register int i = x ; i <= n ; i += lowbit( i ) )
        t[i] += y;
}

inline int find( int x )
{
    int ans = 0;
    for( register int i = x ; i ; i -= lowbit( i ) )
        ans += t[i];
    return ans;
}
```

---

# 树状数组套平衡树

- 维护一个序列支持:
- 1.把x位置的值改为y
- 2.查询一个区间中小于y的数个数

```
inline void insert( int x , int y )
{
    for( register int i = x ; i <= m ; i += lowbit( i ) )
        insert( y , root[i] );
}
```

```
inline int find( int x , int y )
{
    int ans = 0;
    for( register int i = x ; i ; i -= lowbit( i ) )
        ans += rank( y + 1 , root[i] );
    return ans;
}
```



# 区别

- 普通树状数组用到的数据结构：支持修改值，查询值——变量
- 所以普通的树状数组可以用一个数组来维护
- 树状数组套平衡树用到的数据结构：支持插入一个值，查询小于一个值的数个数——平衡树
- 所以树状数组套平衡树可以用一个平衡树的数组维护

# 区别

- 变量每次访问是 $O(1)$ 的，所以树状数组时间复杂度是 $O(\log n)$
- 平衡树每次访问是 $O(\log \text{size})$ 的，所以树状数组套平衡树的最坏时间复杂度是 $O(\log^2 n)$ 的
- 由于不是每棵平衡树都满，所以这里带一个小常数，然而平衡树常数比较大，所以还是比较慢的

所以

- 只需要先写一个平衡树，然后写个树状数组就行了

```

struct Node
{
    int size , value;
    Node * left , * right;
    Node( int s , int v , Node * a , Node * b ) : size( s ) , value( v ) , left( a ) , right( b ) {}
    Node() {}
} * null , ████████████████████ , * root[ MAXN ];

int n , m , cnt;

inline void maintain( register Node * cur )
{
    if( cur -> left -> size > cur -> right -> size * ratio )
        cur -> right = merge( cur -> left -> right , cur -> right ) , st[ --cnt ] = cur -> left , cur -> left = cur -> left -> left;
    else if( cur -> right -> size > cur -> left -> size * ratio )
        cur -> left = merge( cur -> left , cur -> right -> left ) , st[ --cnt ] = cur -> right , cur -> right = cur -> right -> right;
}

int rank( int x , Node * cur )
{
    if( cur -> size == 1 ) return x > cur -> value;
    return x > cur -> left -> value ? rank( x , cur -> right ) + cur -> left -> size : rank( x , cur -> left );
}

void insert( int x , Node * cur )
{
    if( cur -> size == 1 )
        cur -> left = new_Node( 1 , min( cur -> value , x ) , null , null ) ,
        cur -> right = new_Node( 1 , max( cur -> value , x ) , null , null );
    else insert( x , x > cur -> left -> value ? cur -> right : cur -> left ) , maintain( cur );
    update( cur );
}

inline void insert( int x , int y )
{
    for( register int i = x ; i <= m ; i += lowbit( i ) )
        insert( y , root[i] );
}

inline int find( int x , int y )
{
    int ans = 0;
    for( register int i = x ; i ; i -= lowbit( i ) )
        ans += rank( y + 1 , root[i] );
    return ans;
}

```

# Expansion

- 树状数组套平衡树在绝大多数场合下都够用了
- 其他常见的还有树状数组套Trie，线段树套平衡树，可以类比树状数组套平衡树的实现方法来实现，即将内部的树看做一个和外部树独立的数据结构

# 对线段树套平衡树的解释

- 维护一个序列
- 1.单点修改
- 2.区间 $[l,r]$ 中  $\leq k$ 的元素个数

# 线段树套平衡树

- 普通的线段树：
- 每个节点维护一个区间的和
- 每次修改更新 $O(\log n)$ 个节点的和
- 每次查询时用 $O(\log n)$ 个不相交节点的和拼出这次询问的区间
- 线段树套平衡树：
- 每个节点维护一个平衡树，表示区间内部的元素排好序后的平衡树
- 每次修改更新 $O(\log n)$ 个节点的平衡树
- 每次查询时用 $O(\log n)$ 个不相交节点中 $\leq k$ 的元素个数拼出这次询问的区间

# Comparison

- 对比一下几种常见的树套树的优缺点



# 树状数组套树

- 优势：好写，常数
- 劣势：只能维护支持差分的信息，如果不能满足差分则基本上不能使用

# 树状数组套树

- 套平衡树：时间复杂度 $O(\log n^2)$ ，空间复杂度 $O(n \log n)$
- 优势：空间
- 劣势：平衡树没有简单的可以在多个平衡树上二分的方法，区间kth这种询问会多一个log（其实可以不多log的，但这个多树二分的科技常数比较大，而且较为复杂，在OI界应该算很不普及吧）

# 树状数组套树

- 套Trie：时间复杂度 $O(\log n \log v)$ ，空间复杂度 $O(n \log n \log v)$
- 优势：可以简单地在多个Trie上二分
- 劣势：空间（可以通过特殊的技巧达到 $O(n \log n)$ 的空间复杂度）

# 线段树套树

- 套平衡树：时间复杂度 $O(\log n^2)$ ，空间复杂度 $O(n \log n)$
- 套Trie：时间复杂度 $O(\log n \log v)$ ，空间复杂度 $O(n \log n \log v)$
- 优点：可以维护不支持差分的信息
- 缺点：相对难写，慢，空间大

# 平衡树套树

- 套平衡树：时间复杂度 $O(\log n^2)$ ，空间复杂度 $O(n \log n)$
- 套Trie：时间复杂度 $O(\log n \log v)$ ，空间复杂度 $O(n \log n \log v)$
- 优点：可以在线支持第一维插入的问题
- 缺点：挺难写，更慢，常数更大
- 注：外层树用clj定义的“重量平衡”的树
- 如treap，替罪羊树， $BB[\alpha]$ Tree，WBLT等

# 树套OVT

- 就是内部套一个排序后的vector
- Instead of 平衡树
- 优点：好写，空间
- 缺点：复杂度
- 还是最好别用这种奇怪的东西吧

# B树套树

- 据说很快

# Luogu3380 二逼平衡树

- 您需要写一种数据结构（可参考题目标题），来维护一个有序数列，其中需要提供以下操作：
- 查询k在区间内的排名
- 查询区间内排名为k的值
- 修改某一位值上的数值
- 查询k在区间内的前驱
- 查询k在区间内的后继
- $n, m \leq 5e4$



# Analysis

- 平衡树中将x改为y，可以通过插入y之后删除x来实现（注意有的写法空树会导致RE，所以建议先插入后删除）
- 于是单点修改可以通过单点插入，单点删除实现

# Analysis

- 查询kth可以通过二分答案后查询rank实现
- 也就是说每次二分一个答案mid, 然后查询区间中小于mid的数来确定继续二分下去的方向
- 查询前驱后继也可以用类似的方法实现

# Solution1

- 发现本质就是个带单点修改的区间kth
- 可以用线段树套平衡树维护!
- 时间 $O(\log n^3)$

# Solution2

- 发现区间kth可以在多个Trie上一起二分来维护
- 可以用线段树套Trie维护!
- 时间 $O(\log n^2)$

# Solution3

- 发现区间kth可以支持减法
- 可以用树状数组套Trie（权值线段树）维护！
- 时间 $O(\log n^2)$ ，很好写，常数也小

# Solution4

- 通过读论文可以发现可以用动态划分树维护!
- 时间 $O((\log n / \log \log n)^2)$ , 理论优越
- 不过 Not practical

# Note

- 这个不断优化的过程和做这种偏序题的我的思路差不多
- 先给出一个任意poly log (比如 $\log^3, \log^4 \dots$ ) 复杂度的通用化的解法, 然后不断根据题目特性特殊化从而降低复杂度

# 四分树， 八分树

- 裸写四分树最坏的复杂度是 $O(n)$ 的
- 可以通过一些方式进行优化， 不过这里就不介绍了



# 高维分治

- 对于d维的情况
- 时间复杂度 $O(\log n^d)$ , 空间复杂度 $O(n)$
- 优点: 空间, 常数
- 缺点: 对于有的问题会很难写, 强制在线见祖宗

# K-D Tree

- 对于d维的情况
- 时间复杂度 $O(n^{1-1/d} + \log n)$ , 空间复杂度 $O(n)$
- 优点: 空间, 可以支持高维空间打标记
- 缺点: 复杂度, 常数 (在不能剪枝的情况下)

# K-D Tree

- KDT的优势在于可以剪枝和乱搞，如果不能剪枝的话效率很差
- 平面
- 1.在一个位置插入一个数
- 2.矩形和
- 这个问题kdt效率很差
  
- 平面
- 1.在一个位置插入一个数
- 2.矩形max
- 这个问题kdt效率很好

# 如何做高维正交范围类的题呢

- 和上面说的一样
- 反正我是先推出一个poly log的数据结构做法
- 然后想办法去log，到一个可以接受的复杂度

# 如何去log

- 1.降维
- 2.题目有特殊限制
- 3.在数据结构上进行二分

# 降维

- 可持久化
- 离线

# 可持久化

- 就拿静态区间kth这个问题来说
- 刚才讲过
- 显然可以用一个二维数据结构——树套树来维护
- 我决定用线段树套平衡树!
- 好!
- 现在我有了一个单次查询 $O(\log n^3)$ 的优（抠）秀（脚）做法

# 可持久化

- 然后可以发现
- kth是支持在数据结构上二分的信息
- 换成线段树套Trie
- 得到了一个单次查询 $O(\log n^2)$ 的优（抠）秀（脚）做法



# 可持久化

- 然后可以发现，这个问题不支持修改，所以没有必要用树套树，可以换用可持久化Trie（主席树）来解决
- 时间复杂度 $O(\log n)$  单次
- 空间复杂度 $O(n \log n)$  总

# Technology

- 可以查论文，复杂度还可以优化

# 可持久化

- 换个麻烦点的问题:

# Luogu4396 [AHOI2013]作业（加强版）

- 查询区间 $[l,r]$ 内，值域在 $[a,b]$ 内，有多少不同的数
- 强制在线， $n,m \leq 1e6$ ，2GB，8s

# 如何维护区间中不同的数个数

- 升一维，记录每个数上一次出现的位置
- （好像见过的所有维护区间中不同数个数的poly log题都是用的这个东西，无一例外）

# 分析这个问题

- 查询区间 $[l,r]$ 内，值域在 $[a,b]$ 内，有多少不同的数
- 第一维                  第二维                  第三维

- 所以这个题就是个裸的三维偏序维护问题

# 三维？

- 那岂不是 $O(\log n^3)$ 时间单次？
- 发现这个问题其实还是一个静态问题，所以可以通过可持久化降维
- 对于二维问题用可持久化线段树可以维护
- 对于三维问题用可持久化树套树可以维护

# 喵

- 可持久化线段树：拿一维来可持久化，维护这一维前缀的Trie（权值线段树），查询一个前缀中小于 $x$ 的数个数，支持减法从而可以查询区间中小于 $x$ 的数个数
- 可持久化树套树：拿一维来可持久化，维护这一维前缀的树套树，查询一个前缀中第一维小于 $x$ ，第二维小于 $y$ 的数个数支持减法从而可以查询区间中第一维小于 $x$ ，第二维小于 $y$ 的数个数



# 可持久化

- 所以可持久化就是在静态问题上可以降低一维

# 离线

- 1.分治代替可持久化
- 2.其他离线降维方法

# 分治

- 即用分治代替可持久化从而降维
- 一般被称作“整体二分”，“cdq分治”

# Luogu3810 【模板】 三维偏序（陌上花开）

有  $n$  个元素，第  $i$  个元素有  $a_i, b_i, c_i$  三个属性，设  $f(i)$  表示满足  $a_j \leq a_i$  且  $b_j \leq b_i$  且  $c_j \leq c_i$  且  $j \neq i$  的  $j$  的数量。

对于  $d \in [0, n)$ ，求  $f(i) = d$  的数量。

# “CDQ分治”

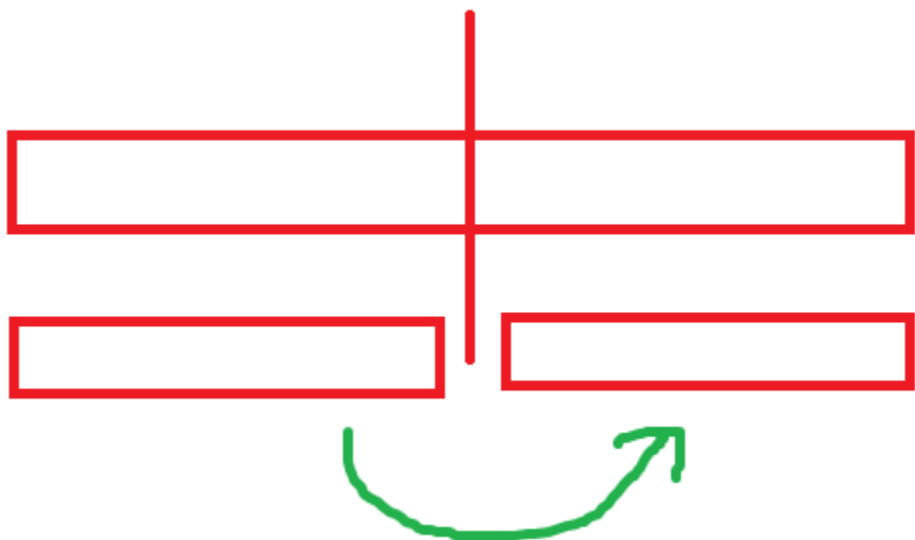
- 考虑如何用分治方法解决三维偏序

# “CDQ分治”

- 先考虑二维偏序也就是 $a_j \leq a_i, b_j \leq b_i$ 怎么做
- 先按 $a$ 为第一关键字， $b$ 为第二关键字排序，那么我们就保证了第一维 $a$ 的有序。
- 于是，对于每一个 $i$ ，只可能1到 $i-1$ 的元素会对它有贡献，那么直接查1到 $i-1$ 的元素中满足 $b_j \leq b_i$ 的元素个数。
- 具体实现？动态维护 $b$ 的树状数组，从前到后扫一遍好啦， $O(n \log n)$ 。
- 那么三维偏序呢？我们只有在保证前两维都满足的情况下才能计算答案了。

# “CDQ分治”

- “CDQ分治”的主要过程就是如果有一维，只有前面的对后面的有贡献，我们可以分治这一维
- 每次分治时计算左边对右边的贡献，然后递归下去
- 这样一直递归到叶子，会发现所有贡献计算完毕



# “CDQ分治”

- 仍然按a为第一关键字，b为第二关键字，c为第三关键字排序，第一维保证左边小于等于右边了。
- 为了保证第二维也是左边小于等于右边，我们还需要排序。
- 考虑类似归并排序的过程中，统计出在子问题中产生的对答案贡献



# “CDQ分治”

- 现在有一个序列，我们把它递归分成两个子问题，子问题进行完归并排序，已经保证 $b$ 有序。此时，两个子问题间有一个分界线，原来第一维左边小于等于右边，所以现在分界线左边的任意一个的 $a$ 当然还是都小于右边的任意一个。只有分界线左边的能对右边的产生贡献。

# “CDQ分治”

- 于是，问题降到了二维。
- 发现是以 $1\log$ 的代价使得维数减少了一维，每层分治时需要计算左边对右边的贡献。
- 这样每层变成一个二维问题。
- 我们可以归并排序（左边的指针为 $j$ ，右边的为 $i$ ）并维护 $c$ 的树状数组，如果当前 $b_j \leq b_i$ ，说明 $j$ 可以对后面加入的满足 $c_j \leq c_i$ 的 $i$ 产生贡献了，把 $c_j$ 加入树状数组；否则，因为后面加入的 $j$ 都不会对 $i$ 产生贡献了，所以就要统计之前被给的所有贡献了，查询树状数组 $c_i$ 的前缀和。
- 总时间复杂度 $O(n \log^2 n)$

# “CDQ分治”

- 总结：
- “CDQ分治”就是指在一维上只有前面对后面有贡献，比如数据结构只有时间前面的修改对时间后面的询问有贡献，也可以使用这个算法，然后分治这一维，分治的每层需要维护分治中线左边对右边的贡献。
- 分治的复杂度： $T(n)=2T(n/2)+O(n)=O(n\log n)$
- 由于问题的转换方法都是一样的，后面的题统一使用树套树的解法。

# Luogu4054 [JSOI2009]计数问题

- 一个 $n*m$ 的方格，初始时每个格子有一个整数权值。接下来每次有2种操作：
  - 改变一个格子的权值；
  - 求一个子矩阵中某种特定权值出现的个数。
- 
- $n, m \leq 300$ , 值  $\leq 100$ ,  $q \leq 2e5$

# Solution

- 翻译：有 $n*m$ 个点，每个点有三个权值 $x_i$ ,  $y_i$ ,  $z_i$
- 1. 把一个点的 $z_i$ 进行修改
- 2. 查询有多少点满足
- $l_1 \leq x_i \leq r_1$
- $l_2 \leq y_i \leq r_2$
- $l_3 = z_i$

# Solution

- 可以发现，这个是一个三维的偏序维护问题
- 有一个 $O(\log n^3)$ 单次的方法

# Solution

- 可以发现 $x_i$ ,  $y_i$ 只有300,  $z_i$ 只有100, 所以可以用三维树状数组维护
- 然后发现 $z_i$ 这一维只查询 $[z_i = l3]$ 的所有点
- 所以可以通过开100个二维树状数组来去掉 $z_i$ 这一维
- 于是就变成了一个单点修改, 矩形和的题了
- 时间复杂度 $O(\log n^2)$ 单次
- $(x, y, a) \rightarrow (x, y, b)$ 就在 $a$ 的二维树状数组上 $(x, y)$ 位置-1,  $b$ 的二维树状数组上 $(x, y)$ 位置+1

# 动态逆序对

- Luogu3759 [TJOI2017]不勤劳的图书管理员
- Luogu3157 [CQOI2011]动态逆序对
- 即带修改，维护全局逆序对对数



# Solution

- 翻译：有 $n$ 个点，每个点两个属性 $a_i, b_i$
- 每次修改一个点的 $b_i$ ，维护有多少点对 $(i, j)$ 满足 $[a_i < a_j \&\& b_i > b_j]$

# Solution

- 考虑每次修改的时候，对答案产生的贡献
- 假设原来是 $(a_i, b_i)$ 变为 $(a_i, b_i')$
- 发现只对和被修改点有关的点对造成影响
- 贡献为减去所有和 $(a_i, b_i)$ 有贡献的，加上所有和 $(a_i, b_i')$ 有贡献的点对个数

# Solution

- 减去：第一维 $<a_i$ ，第二维 $>b_i$ 的，第一维 $>a_i$ ，第二维 $<b_i$ 的
- 加上：第一维 $<a_i$ ，第二维 $>b_i'$ 的，第一维 $>a_i$ ，第二维 $<b_i'$ 的
- 加上的个数和减去的个数都是一个矩形数点
- 于是用树套树可以轻松维护了
- $O(\log^2 n)$

# Luogu3332 [ZJOI2013]K大数查询

- 1.区间 $[l,r]$ 插入 $x$
- (这里的插入指的是在每个位置上面放上一个新的数, 不是每个数旁边放一个, 可以看作每个位置是一个vector, 在区间push\_back了一下 $x$ )
- 2.区间kth

# Solution

- 翻译：有 $n$ 个点，每个点有权值 $a_i, b_i$
- 1. 加入 $(l_1, x), (l_1 + 1, x) \cdots (r_1, x)$ 这些点
- 2. 求满足 $l_1 \leq a_i \leq r_1$ 的所有点中 $b_i$ 的 $k$ th

# Solution

- 怎么做呢
- 按照套路我们先建一个树套树
- 但是发现一个问题：区间插入的操作无法高效地在外层树上实现，因为无法下放

# Solution

- 发现区间插入这个操作不好维护，可以考虑把位置和权值两维交换一下
- 于是从加入 $(l1, x), (l1+1, x) \cdots (r1, x)$ 这些点
- 变为了在 $x$ 位置加入 $[l1, r1]$ 这些点
- 从一个树套树的外层树区间修改，内层树单点修改
- 变为了一个树套树外层树单点修改，内层树区间修改

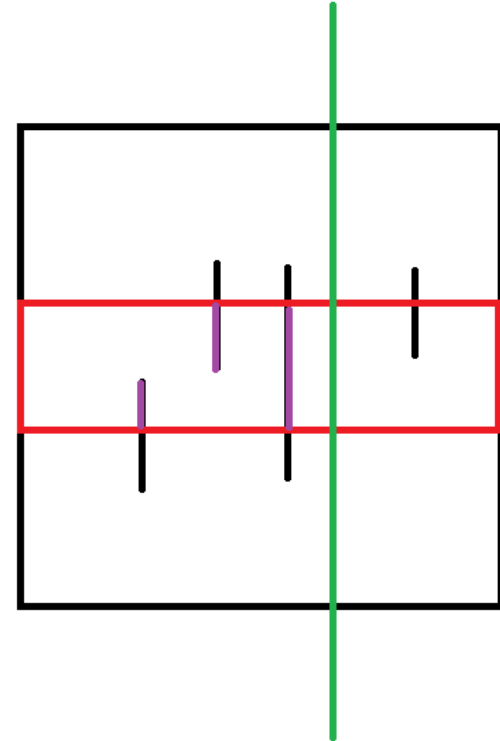
# Solution

- 现在有了一个时间复杂度 $O(\log n^3)$ 单次的做法:
- 每次二分答案一个 $\log$
- 树套树两个 $\log$



# Solution

- 可以做到时间复杂度 $O(\log n^2)$ ，在外层的线段树（或者树状数组）上二分即可
- 即找到一个点，满足左边的区间在 $[l, r]$ 中的元素和为 $k$
- 当然，这个题不强制在线，也可以通过各种分治来做



# Notice

- 其实外部维护区间也是可以的，因为偏序问题基本上都是对称的，按什么顺序维护都是可以在同样的复杂度维护的
- 但是这个涉及到一个好不好写的问题…所以如果发现直接做比较麻烦可以试着换换偏序维护的顺序

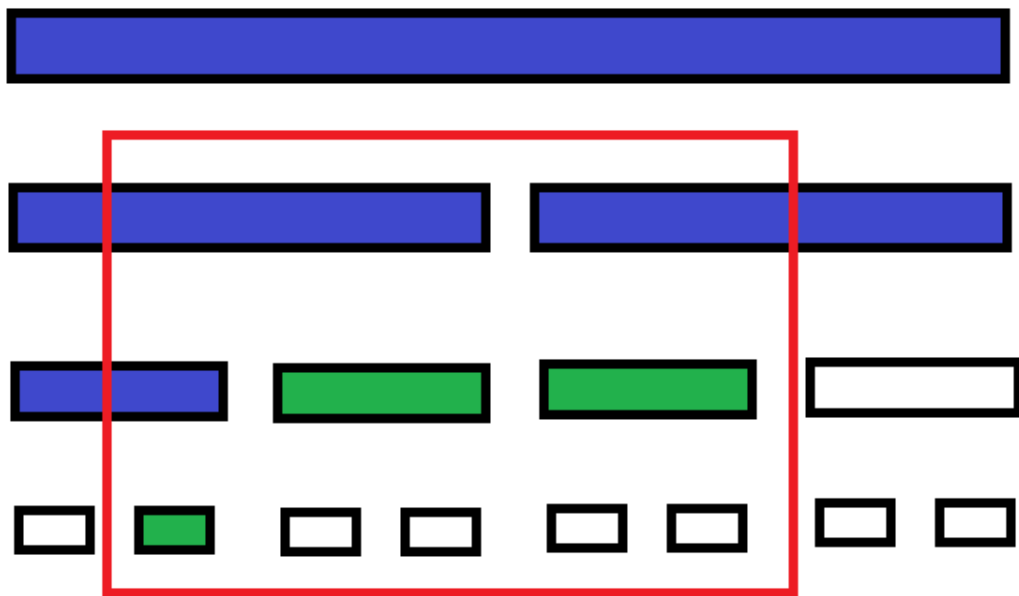
# Solution

- 对序列建立线段树，内部套一棵平衡树
- 注意到树套树的结构一般是不支持标记下放的
- 考虑使用标记永久化的方法

# Solution

- 外层线段树上每个点开一个trie树或值域线段树，维护这个点上被完整的插入的元素
- 再开一个trie树或值域线段树，维护这个点子树内部所有元素（这里元素个数可能非常多，但一个 $x$ 出现 $y$ 次这样的实际上可以一起处理）
- 每次查询时即外层线段树上 $O(\log n)$ 个节点，其子树内部的所有元素构成的trie，与其上方虚树构成的trie上一起二分

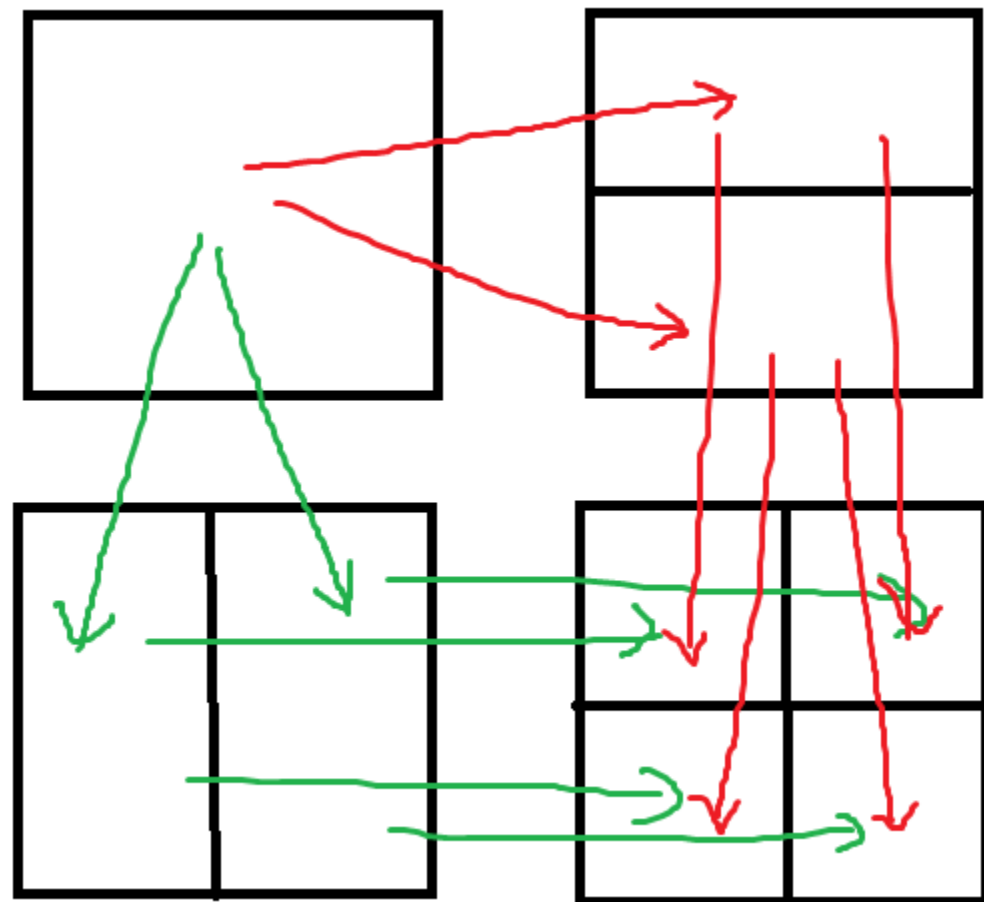
# Solution



- 其中绿色子树内的所有元素对答案有贡献
- 蓝色部分也对答案有贡献，每个蓝色部分贡献需要乘的数不一样

# 树套树为什么难以支持标记下放

- 因为树套树不是树，是DAG结构
- DAG结构一般不能支持标记下放
- 类似的有带区间复制的可持久化平衡树



# Luogu4278 帶插入區間K小值

- 維護一個序列，支持在一個位置插入，刪除，或查詢區間k小值
- 強制在線

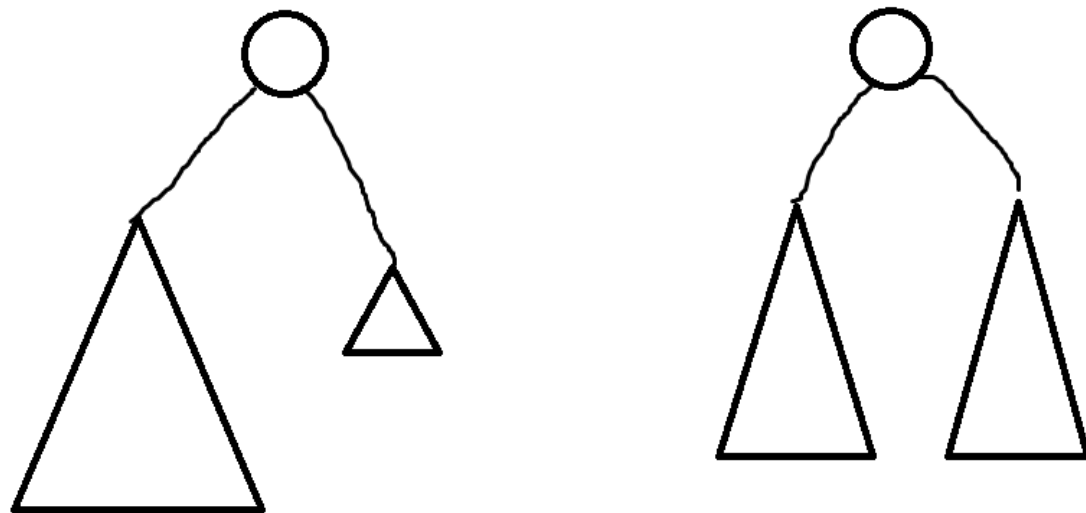
# Solution

- 如果带修改我们都会做，但强制在线带插入怎么办呢？
- 可以在外层使用“重量平衡树”
- 定义为经过 $n$ 次操作后，所有旋转的节点或者重构的节点的size的和为 $O(n \log n)$
- 代表有替罪羊树，treap，WBLT



# Solution

- 替罪羊树：让每个节点每个儿子占size的比例不会特别大
- 设定常数 $\alpha$ 表示最大的一个儿子占整体的size比例最大为多少
- 如果一个节点这个比例超过 $\alpha$ ，则暴力重构这个节点，重构的表现为DFS子树，得到中序遍历，之后像线段树建树那样建立一棵完美平衡的子树，然后塞回来



# Solution

- 可以证明替罪羊树重构的总复杂度为 $O(n \log n)$
- 用类似势能的方法来证
- 设定 $f(x) = \max(x.\text{left.size}, x.\text{right.size}) - \min(x.\text{left.size}, x.\text{right.size})$
- 因为替罪羊树保证了那个 $\alpha$ 的性质，所以深度为 $O(\log n)$
- 所以每次插入只会导致 $O(\log n)$ 个节点的 $f(x)$ 发生变化，每个 $f(x)$ 最多增加1，所以总共最多增加 $O(\log n)$
- 每次重构时，重构代价为 $x.\text{size}$ ，此时 $f(x) = \alpha * x.\text{size} - (1 - \alpha) * x.\text{size}$ ，故减少了 $x.\text{size}$ 的一个常数比例

# Solution

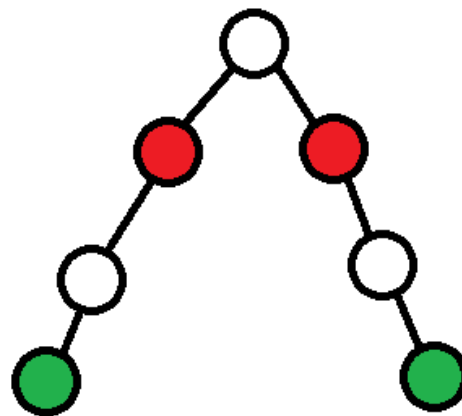
- 于是这道题使用“重量平衡树”套一个trie（值域线段树）即可
- 总时间复杂度 $O((n+m)\log^2 n)$ ，空间也为同复杂度，可以使用压缩trie的方法优化空间
- 小心被分块之类的吊打，树套树空间很大的时候一般都很不妙

# Luogu3242 [HNOI2015]接水果

- 树，先给了一些路径，每个路径有个权值
- 然后每次查询给一个路径，求包含这个路径的最开始给的路径中，权值kth是多少？
- Bonus：如果是查询这个路径包含的初始路径里面权值kth呢？

# 转化

- 定义 $i$ 点在树的DFS序的开始时间戳为 $li$ ，结束时间戳为 $ri$
- 如果 $lx \leq ly \leq ry \leq rx$ 则 $y$ 为 $x$ 祖先
- 于是一个路径 $(x,y)$ 的所有子路径 $(u,v)$ 可以被定义为满足 $lx \leq lu \leq rx, ly \leq lv \leq ry$ 的所有点
- （当然还有 $x$ 是 $y$ 祖先的情况，不过这个这个差不多，就不详细说了）



# Solution

- 问题即变为多次查询矩形内的kth
- 使用可持久化树套树即可维护
- 第二个问呢?

# 转化

- 这种路径包含问题，一般都是把其左右端点的DFS序当作两个属性 $a_i$ 和 $b_i$ 来处理
- 求出原树的DFS序，即最开始给的路径可以被定义为一个矩形，矩形中每个位置都插入一个权值

# 转化

- 即开始给出 $n$ 个矩形, 每个矩形里面插入一个数
- 每次查询单点的 $k$ th



# Solution

- 扫描线扫一维，变成区间插入，区间删除，单点kth，就转化为那个k大数查询问题了
- 这里强制在线，所以我们使用可持久化树套树来维护

# P9068 [Ynoi Easy Round 2022] 超人机械 TEST\_95

给定一个序列  $a$ ，我们定义一个二元组  $(i, j)$  为一个逆序对当且仅当  $i < j$  且  $a_i > a_j$ 。定义两个逆序对  $(i_1, j_1), (i_2, j_2)$  **本质不同** 当且仅当  $a_{i_1} \neq a_{i_2}$  或  $a_{j_1} \neq a_{j_2}$ 。

现在给出  $a$  序列，问本质不同逆序对个数。

这还不够。

现在有  $q$  组修改，每一次修改形如  $x\ y$  表示修改  $a_x$  为  $y$ ，每一次修改 **不互相独立**，即这一次修改会影响到后面的所有修改。

你需要对于每一次修改输出序列本质不同逆序对个数。

# Solution

维护两个序列  $a', b'$ ，分别维护每个元素第一次出现位置，以及每个元素最后一次出现位置，如果一个位置不满足条件，则不在对应序列中存在。

然后对每个  $a'$  中存在的  $i$ ，以及每个  $b'$  中存在的  $j$ ，如果  $a'[i] > b'[j]$  且  $i < j$ ，则构成逆序对，相当于我们统计这两个序列之间的逆序对，可以证明这个和问题中的逆序对是一一对应的。

于是使用和单点修改，全局逆序对类似的方法，将问题转换为单点修改，查询矩形点数即可。

总时间复杂度  $O(n \log n + m \log^2 n)$ 。

# Luogu 4690 [Ynoi2016]镜中的昆虫（弱化版）

- 维护一个长为  $n$  的序列  $a[i]$ ，有  $m$  次操作。
- 1.将  $x$  位置修改为  $y$ 。
- 2.询问区间  $[l,r]$  出现了多少种不同的数。
- 也就是说同一个数出现多次只算一个。

# Solution

# Luogu 4690 [Ynoi2016]镜中的昆虫

- 维护一个长为  $n$  的序列  $a[i]$ , 有  $m$  次操作。
- 1.将区间  $[l,r]$  的值修改为  $x$ 。
- 2.询问区间  $[l,r]$  出现了多少种不同的数。
- 也就是说同一个数出现多次只算一个。

# Solution

- 这个是个裸的区间染色段数均摊
- 考虑维护每个点与其颜色相同的前驱
- 用平衡树维护颜色连续段，然后考虑一个颜色连续段里面除了第一个点以外，后面的点 $i$ 的前驱都是 $i-1$
- 用树套树/分治维护，于是这个颜色段的变化可以 $O(\log^2 n)$ 维护
- 总复杂度 $O((n+m)\log^2 n)$
  
- 注意常数...
- 小心细节...

# Bz oj 3489 (加强版)

- 给出一个长度为 $n$ 的序列，给出 $m$ 个询问：在 $[l,r]$ 之间找到一个在这个区间里只出现过一次的数，并且要求找的这个数尽可能大。如果找不到这样的数，则直接输出0。
- $n \leq 1e5, m \leq 1e6, 256MB$
- 强制在线



# Solution

- 还是通过前驱后继来算出每个位置的贡献范围
- 设 $i$ 的前驱为 $p_i$ ，后继为 $q_i$ ，把询问看成点 $(L,R)$ ，有贡献的 $i$ 满足 $L \in (p_i, i]$ 且 $R \in [i, q_i)$ ，询问的就是覆盖这个点的矩形的最大值。

# Solution

- 那么可以用可持久化树套堆，插入矩形时一维可持久化，一维区间插入，用堆维护最大值。
- 注意这里的“可持久化堆”只需要查询历史，因此只需要把最大值记下来就好了。
- 因为这里“可持久化堆”询问无自由度，所以部分可持久化的堆和可持久化的变量完全等价。
- 总时间复杂度 $O(n \log^2 n + m \log n)$ ，空间复杂度 $O(n \log n)$

# Luogu7476 「C.E.L.U-02」 苦涩

YQH 的脑中可以被分成  $n$  个片区，每个片区相当于一个存放记忆的可重集，初始为空。他将进行  $m$  次这三种操作：

操作 1：区间  $l \sim r$  的片区中都浮现了一个苦涩值为  $k$  的记忆。

操作 2：YQH 开始清理  $l \sim r$  片区的记忆。如果一个片区  $k \in [l, r]$  且  $k$  中苦涩值最大的记忆与  $l \sim r$  片区中苦涩值最大的记忆相等，则将这个苦涩值最大的记忆忘记。如果在同一个片区有多个相同的苦涩值最大的记忆，则只忘记一个。如果这些片区内没有记忆，则无视。

操作 3：YQH 想知道， $l \sim r$  片区中苦涩值最大的记忆的苦涩值是多少，如果不存在，输出 `-1`。

# Solution

# 2021牛客暑期多校训练营6 E Growing Tree

## • 题意

给出一棵树，最开始只有一个根结点，编号为 1，有颜色  $c$ 。要求 **在线** 完成  $m$  次操作，每次操作是以下两种之一

1. 给结点  $u$  加一个叶结点  $n + 1$ ，颜色为  $c$ 。
2. 询问以  $u$  为根的子树中颜色为  $c$  的结点的个数。

操作数  $m \leq 5 \times 10^5$ 。

# Solution

# P5445 [APIO2019] 路灯

一辆自动驾驶的出租车正在 Innopolis 的街道上行驶。该街道上有  $n + 1$  个停车站点，它们将街道划分成了  $n$  条路段。每一路段都拥有一个路灯。当第  $i$  个路灯亮起，它将照亮连接第  $i$  与第  $i + 1$  个站点的路段。否则这条路段将是黑暗的。

安全起见，出租车只能在被照亮的路段上行驶。换言之，出租车能从站点  $a$  出发到达站点  $b$  ( $a < b$ ) 的条件是：连接站点  $a$  与  $a + 1$ ,  $a + 2$  与  $a + 3$ , ...,  $b - 1$  与  $b$  的路段都被照亮。

在经过一些意外故障或修理之后，街道上的路灯可能是亮起的，也可能是熄灭的。

现在给定 0 时刻时，街道上路灯的初始状态。之后  $1, 2, \dots, q$  时刻，每时刻会发生下列两种事件之一：

- toggle  $i$ : 切换第  $i$  个路灯的状态。具体地说，若路灯原来亮起，则现在将熄灭；若路灯原来熄灭，则现在将亮起。
- query  $a\ b$ : 出租车部门的负责人想知道，从 0 时刻起到当前时刻，有多少个时刻满足：出租车能够从站点  $a$  出发到达站点  $b$ 。

请你帮助出租车部门的负责人回答他们的问题。

# Solution

- 用 $(x,y)$ 表示从 $x$ 走到 $y$ 的答案
- 使用set维护所有亮灯的连续段
- 每次连续段的变化是 $O(1)$ 的
- 对于点 $x$ ，设 $x$ 所在的极长亮灯段左边端点为 $l$ ， $x+1$ 右边端点为 $r$
- 假设 $[x,x+1]$ 的路灯开了，那么 $[l,x]$ 会与 $[x+1,r]$ 合并
- 考虑差分贡献，设当前时刻为 $t$ ，合并时对 $[l,x] \times [x+1,r]$ 这个矩形加上 $m-t$ ，即先假设他们一直连通到最后时刻



# Solution

- 假设 $[x, x+1]$ 的路灯关了, 那么 $[l, x]$ 会与 $[x+1, r]$ 分离, 则我们将 $[l, x] \times [x+1, r]$ 的矩形减去 $m-t$
- 发现这样差分计算出了每个点维持现状到结束时的答案总和
- 比如如果一个位置在 $t_1$ 时连通,  $t_2$ 时断开
- 则其被加了 $(m-t_1)-(m-t_2)=t_2-t_1$ , 刚好为我们想要的答案

# Solution

- 查询时进行单点查询即可
- 如果查询时两点连通，答案需要减去 $(m-t)$
- 使用树套树维护矩形加单点查
- $O(m \log^2 n)$

# Luogu3688 [ZJOI2017]树状数组

- 题目过长，大家去洛谷上看吧

# Solution

- 推一下性质可以发现可怜求的是后缀和
- 设数列为 $A$ , 那么可怜求的就是 $A[l-1]$ 到 $A[r-1]$ 的和（即 $l-1$ 的后缀减 $r$ 的后缀）,而答案为 $A[l]$ 到 $A[r]$ 的和
- 这两种答案都包含 $A[l]$ 到 $A[r-1]$ 的和, 因此只需判断 $A[l-1]$ 与 $A[r]$ 相等的概率就行了

# Solution

- 考虑记下每次修改的影响，假设已知左端点 $a$ 和右端点 $b$ ，那么对于某一次修改区间 $[l, r]$ ，则只有当 $a \in [l, r]$ 或 $b \in [l, r]$ 时才有影响

# Solution

- 设 $p$ 为任选区间内一个数的概率，这里分三种情况讨论：
- $a \in [1, l-1], b \in [l, r]$ 时，有 $1-p$ 的概率不影响
- $a \in [l, r], b \in [l, r]$ 时，有 $1-2p$ 的概率不影响
- $a \in [l, r], b \in [r+1, n]$ 时，有 $1-p$ 的概率不影响
- 那么只要把所有的影响都合并起来就行了，设当前相同概率为 $p$ ，当前修改不影响的概率 $q$ ，则相同概率更新为 $p*q + (1-p)*(1-q)$

# Solution

- 考虑用二维点对 $(x,y)$ 表示 $A[x]$ 和 $A[y]$ 相等的概率
  - 可以发现每次修改就是 $O(1)$ 次矩形乘一个数
  - 查询就是查询单点值
  - 所以用一个树套树，矩形打永久化标记，查询单点信息就行了
  - 注意到这里可能没有逆元，所以不能用树状数组套树来维护，需要用线段树套树来维护
- 
- $O(m \log^2 n)$

# 内层树维护标记

- 有的时候有复杂的区间修改操作则需要打标记
- 因为树套树是DAG而不是树，所以有两种常见的方法：
  - 1.内层维护标记永久化的树
  - 2.内层使用DAG数据结构，比如可持久化平衡树，这样可以标记下放



# 经典问题

- 维护一个 $n \times n$ 的二维平面，每个位置初始为0
- 有 $m$ 次操作：
  - 1. 矩形对 $x$ 取max
  - 2. 查矩形max
- 强制在线

# Solution

- 考虑使用树套树维护，第一维分治 $x$ ，第二维分治 $y$
- 维护两个内层树，第一个表示所有子树内的标记的影响，第二个表示所有这个外层树节点上的标记的影响
- 矩形修改的时候我们直接标记永久化，考虑第一维修改的 $O(\log n)$ 个线段树节点的虚树，我们在这 $O(\log n)$ 个外层树节点的内层树上进行区间 $\max$ 操作
- 询问的时候需要查询 $O(\log n)$ 个外层树节点的内层树区间 $\max$

# Solution

- 询问的时候再把这外层树节点的虚树上每个查一下对应内层树区间的max
- 这样我们维护了子树内的贡献，维护了子树上层的贡献，所以正确性有保证
- 矩形加矩形和也能这么做，不过有简单些的做法
- 总时间复杂度 $O(n\log n + m\log^2 n)$

# Luogu2086 [NOI2012] 魔幻棋盘

将要读二年级的小 Q 买了一款新型益智玩具——魔幻棋盘，它是一个  $N$  行  $M$  列的网格棋盘，每个格子中均有一个正整数。棋盘守护者在棋盘的第  $X$  行第  $Y$  列（行与列均从 1 开始编号）并且始终不会移动。棋盘守护者会进行两种操作：

- (a) 询问：他会以自己所在位置为基础，向四周随机扩展出一块大小不定的矩形区域，向你询问这一区域内所有数的最大公约数是多少。
- (b) 修改：他会随意挑选棋盘上的一块矩形区域，将这一区域内的所有数同时加上一个给定的整数。

游戏说明书上附有这样一句话“聪明的小朋友，当你连续答对 19930324 次询问后会得到一个惊喜噢！”。小 Q 十分想得到这个惊喜，于是每天都在玩这个玩具。但由于他粗心大意，经常算错数，难以达到这个目标。于是他来向你寻求帮助，希望你帮他写一个程序来回答棋盘守护者的询问，并保证 100% 的正确率。

为了简化问题，你的程序只需要完成棋盘守护者的  $T$  次操作，并且问题保证任何时刻棋盘上的数字均为不超过  $2^{62} - 1$  的正整数。

# Solution

# Bzoj3946 (改)

- 维护一个长为 $n$ 的序列，每个位置是个长度 $\leq 10$ 的字符串，有 $m$ 次操作：
- 1.给区间字符串前面加上一个输入的字符串 $S$
- 2.求区间lcp长度
- 我编个看上去合适的数据范围 $n \leq 1e6, m \leq 3e4, \sum |S| \leq 1e7$
- 强制在线

# Solution1

- 第一维用线段树分治掉，套一个可持久化平衡树维护标记，一个可持久化平衡树维护区间lcp
- pushdown的时候标记的可持久化平衡树直接区间复制合并到儿子的标记的可持久化平衡树上，同时把标记的可持久化平衡树区间复制合并到节点的答案可持久化平衡树上
- lcp直接在可持久化平衡树上二分
- 总时间复杂度 $O(n+m\log^3 n+|S|)$

# Solution2

- 考虑利用差分的思想，维护相邻两个位置的 LCP 长度
- $f[i]$  表示  $i$  和  $i+1$  位置的字符串的 LCP 长度
- 询问区间  $[l, r]$  的 LCP 长度，可以表示为  $f[l] \sim f[r-1]$  的区间  $\min$
- 区间 `push_front` 的时候只影响边界位置的  $f$
- 还是沿用线段树套可持久化平衡树的方法，每次只需要在两个位置求两棵可持久化平衡树维护的字符串的 LCP，二分哈希即可
- 总时间复杂度  $O(n + m \log^2 n + |S|)$



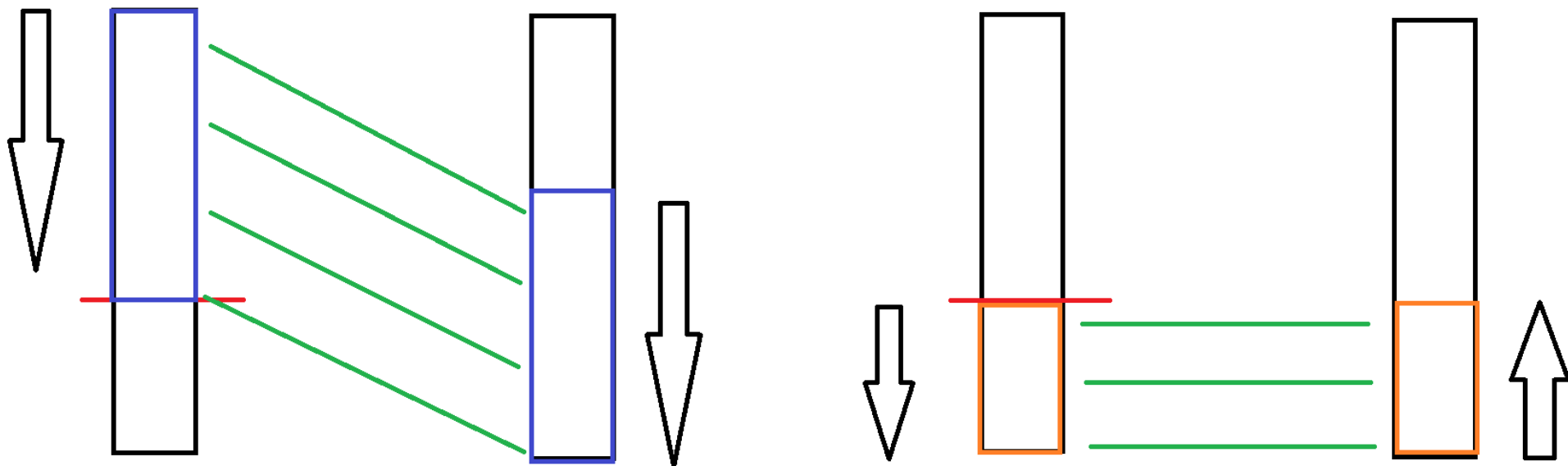
# HBCPC2021 K

- 给定一个序列 $a$ ，每次查询时给出 $l, r, v$
  - 令 $i$ 从 $l$ 到 $r$ ，依次做 $v = |v - a_i|$ 的迭代，求最后的 $v$
  - 强制在线
- 
- $n, m \leq 1e5$ , 值域为 $[0, 1e9]$
  - 时间限制：6s，空间限制：2G

# Solution

这个问题的分段函数段数很大，只需  $O(\log v)$  个序列上的位置就可以构造出  $\Theta(v)$  段的分段函数，所以需要使用一种 DAG 结构的数据结构，比如树套树或者带区间复制的可持久化平衡树。

先考虑全局查询怎么做，可以从右往左扫描线，维护一个可持久化平衡树，初始为恒等映射，左边加入一个线性变换  $x = |x - a_i|$  时，等价于将所有左边  $\geq a_i$  的元素  $x'$  会映射为  $x' - a_i$ ，所有  $< a_i$  的元素  $x'$  会映射为  $a_i - x'$ 。于是等价于将右边的映射中一个区间复制过来，另一个区间翻转后复制过来，然后合并，这个使用可持久化平衡树可以高效地维护。



# Solution

为了处理区间查询，只需将序列进行一次分治即可，最后的数据结构是一个线段树套可持久化平衡树的结构。

std有进行一些常数优化，如线段树最底层按  $\Theta(\log n \log v)$  大小分块，线段树最上层去掉  $O(\log \log n)$  个节点，改为分块，线段树可以直接从右儿子中插入左子树部分的操作而得到父亲结点的可持久化平衡树。

总时间复杂度  $O((n + m) \log n \log v)$ 。

# CF1340F Nastya and CBS

你需要动态维护一个多种括号组成的括号序列。需要支持两种操作：

- $1\ i\ t$ : 修改  $i$  位置的括号为  $t$  (绝对值表示种类, 正数表示是左括号, 负数表示是右括号)。
- $2\ l\ r$ : 查询  $[l, r]$  区间是否是一个合法的括号序列。

序列长度为  $n$ , 不同的括号种类数为  $k$ , 操作次数为  $q$ 。

一个多种括号组成的括号序列  $S$  是合法的当且仅当：

1.  $S$  为空。
2.  $S$  可以表示为  $A + B$  的形式, 其中  $A, B$  都是合法的。
3.  $S$  可以表示为  $(^* + A + )^*$  的形式, 其中  $A$  是合法的并且  $(^*$  与  $)^*$  是一对相同种类的左右括号。

例如 `0{0}0` 和 `{({})}` 是合法的, 而 `1[1]` 和 `{({})}` 不是。

# Solution1

- 考虑线段树维护序列， 对一个节点维护其经过括号匹配后的结果， 如果一个节点出现了两个不匹配的左右括号挨在一起的情况（如“[]”）， 则这个节点不可能被成功匹配， 可以直接标记匹配失败
- 所以一个节点如果可能和别的节点合并后成功匹配， 其不匹配括号一定是一些左括号然后一些右括号
- 为了方便判括号是否可以匹配， 使用字符串哈希的方法， 考虑我们对每个节点维护了其不匹配括号中左括号与右括号分别的哈希值， 如何合并节点信息

# Solution1

- 这里用小括号，只表示括号方向
- 如)))(((与)))(合并得到))) (，可以发现合并时，一定会抵消掉左儿子的所有右括号，或者右儿子的所有左括号，否则匹配失败
- 如上述例子合并时，我们需要知道右儿子这x个左括号的哈希值，这个可以通过在右儿子的子树中递归实现：
- 若右儿子的左儿子的左括号 $\geq x$ 个，我们只需要递归右儿子的左儿子即可
- 否则右儿子的左儿子的左括号一定全部包含在这x个括号中，因为维护了所有不匹配左括号的哈希值，所以只需要递归右儿子的右儿子

# Solution1

- 这样合并时构成了一个单侧递归结构，合并两个节点的时间复杂度为 $O(\log n)$ ，总时间复杂度为 $O(\log^2 n)$
  - 建树的时间复杂度为 $T(n) = 2T(n/2) + O(\log n)$ ，解得 $T(n) = O(n)$
  - 总时间复杂度 $O(n + m \log^2 n)$
- 
- P.S 四年前ccz想过区间不匹配括号的维护，他题解给的那个线段树套可持久化平衡树的方法是被完爆的。我们出过区间不匹配括号半群的题P6781，当时设计了一个类top tree结构，实际上也是可以用这个单侧递归方法做的，只是不可减了要麻烦一些

# Solution2

- 可以考虑用线段树套可持久化平衡树维护
- 线段树维护序列，可持久化平衡树维护区间合并后剩下的括号
- 每次合并的时候可以二分，然后进行一次区间复制
- 总时间复杂度 $O(n+m\log^2 n)$ ，空间复杂度 $O(n+m\log n)$



# LOJ6120. 「2017 山东二轮集训 Day7」领主

在某个鬼畜的大陆上，有若干领主，这片大陆可以看做是一个无限平面，领主们会不断试图占领土地，具体来说，有三种行为：

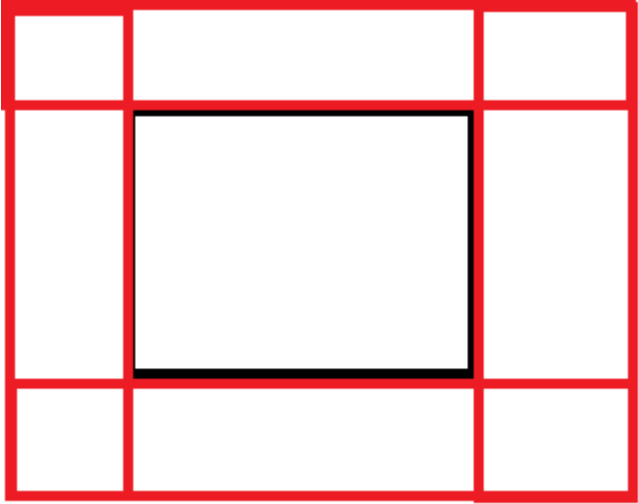
- I  $x_1$   $y_1$   $x_2$   $y_2$  有一个新的领主想要试图占领以  $(x_1, y_1)$  为左上角， $(x_2, y_2)$  为右下角的矩形土地；
- D  $x$  第  $x$  个领主放弃了他的土地占领计划；
- Q  $x_1$   $y_1$   $x_2$   $y_2$  有秘密组织想知道，他们如果用  $(x_1, y_1)$  为左上角， $(x_2, y_2)$  为右下角的矩形土地开展活动，最多会与多少个领主发生冲突（若与某个领主想要占领的土地有重合点，即视为可能发生冲突）。

现在你知道所有的行为信息，需要回答秘密组织的问题。

- 强制在线

# Solution

- 考虑求有多少不相交的
- 然后直接树套树即可



# CF848C Goodbye Souvenir

给定长度为 $n$ 的数组, 定义数字 $X$ 在 $[l, r]$ 内的值为数字 $X$ 在 $[l, r]$ 内最后一次出现位置的下标减去第一次出现位置的下标

给定 $m$ 次询问, 每次询问有三个整数 $a, b, c$ , 询问规则如下:

当 $a = 1$ 时, 将数组内第 $b$ 个元素更改为 $c$

当 $a = 2$ 时, 求区间 $[b, c]$ 所有数字的值的和



# Luogu6749 『MdOI R3』 Yoshino

Yoshino 给了你一个长度为  $n$  的序列，第  $i$  项为  $a_i$ 。

现在 Yoshino 会对数列进行  $m$  次操作。

操作分成两种：

- 1  $l\ r\ x$  Yoshino 把数列下标在  $[l, r]$  区间内的数修改为了一个从  $x$  开始公差为 1 的等差数列。
- 2 Yoshino 需要查询整个数列中的逆序对个数。逆序对的定义为数对  $(i, j)$  满足  $i < j$  且  $a_i > a_j$ 。

# Solution

- 建一棵线段树套平衡树
- 每次区间修改的时候等价于在 $O(\log n)$ 个线段树节点上进行了区间修改
- 对于这次修改完整包含的线段树节点打个赋值标记
- 对于没有完整包含的线段树节点，考虑颜色段均摊，总共在所有节点上进行了 $O((n+m)\log n)$ 次修改，每次修改需要修改其平衡树
- 询问在树套树外层分治后，对应的 $O(\log n)$ 棵平衡树上查就行
- 总时间复杂度 $O((n+m)\log^2 n)$

# Luogu7825 「RdOI R3」 VSQ

设  $x$  为一个长度不小于 2 的 01 串，如果  $x$  的所有长度为 2 的子串的异或和均为 1，则  $x$  为一个「VS 串」。

你需要维护一个长度为  $n$  的 01 串  $a$ ，支持以下操作：

格式	说明
0 l r	将区间 $[l, r]$ 填充为 0
1 l r	将区间 $[l, r]$ 填充为 1
2 l r	将区间 $[l, r]$ 中的每个数取反，即 1 变 0，0 变 1
3 l r k	查询 $[l, r]$ 区间中有多少个长度为 $k$ 的子串是「VS 串」
4 l r	查询 $[l, r]$ 区间内最长的「VS 串」的长度，如果区间内不存在这样的子串则输出 1

# Solution



# [Ynoi????] pmrmscxip

给序列  $a_1, \dots, a_n$  和排列  $b_1, \dots, b_n$ , 共有  $m$  次操作:

修改操作: 给定  $x, y$ , 将  $a_x$  改为  $y$ ;

查询操作: 给定  $l, r, x$ , 查区间  $[l, r]$  内最长的子区间  $[l', r']$  (即满足  $l \leq l' \leq r' \leq r$ ), 使得对  $l' \leq i < r'$  有  $a_{i+1} = b_{a_i}$ , 且存在  $l' \leq i \leq r'$  使得  $a_i = x$ 。需要输出满足条件的  $r' - l' + 1$  的最大值, 若不存在则输出 0。

$$1 \leq n, m \leq 10^6;$$

$$1 \leq a_i \leq n;$$

$$1 \leq b_i \leq n, \text{ } b_i \text{ 互不相同};$$

$$\text{对修改操作, 满足 } 1 \leq x, y \leq n;$$

$$\text{对查询操作, 满足 } 1 \leq l \leq r \leq n, \text{ } 1 \leq x \leq n.$$

# Solution

# 经典问题

# Solution

# Summarization

- 有几个常见的转换：
- 链  $\rightarrow$  两端的DFS序构成的矩形
- $x$ 子树中距离 $x \leq k$ 的点  $\rightarrow$  DFS序一维，深度一维所构成的矩形
- 区间中不同数个数  $\rightarrow$  每个点记下前驱构成的二维点的矩形
- .....

# 一些其他的嵌套结构的问题

- 下面的题目和树套树关系不大，不过也是数据结构之间的嵌套

# Luogu5073 [Ynoi2015]世界で一番幸せな女の子

- 维护序列支持:
- 1.全局加
- 2.区间最大子段和

# Solution1

- 由于可以加负数
- 设 $pre[x]$ 为 $x$ 时刻全局加的值
- 按照 $pre$ 从小到大处理
- 这样就转换为了只加正数



# Solution1

- 考虑分块
- 每块维护一下块内长为 $1 \cdots \sqrt{n}$ 的最大前缀，最大后缀，最大子段和
- 如果一个整块加了，那么这个整块块内的最大子段长度肯定是不降的

# Solution1

- 每次查询的时候按照当前的全局加标记确定出每个块内的最大子段和
- 然后把跨块的询问用左边块的最大后缀和右边块的最大前缀拼起来
- 即可
- 总复杂度 $O( m\sqrt{n} + n\sqrt{n} )$

# Solution2

- 这个根号太屑了，考虑poly log

# Solution2

- 线段树维护：
- 每个节点的左，右，内部最大子段和的凸函数
- 凸函数是一个区间的半平面交，每个下标 $x$ 对应的意义是这个节点被整体加了 $x$ 后，该节点的左，右，内部最大子段和的值是多少
- 设左最大子段和是 $pre$ ，右最大子段和是 $suf$ ，内部最大子段和是 $mid$

# Solution2

- 然后建树的时候可以发现
- $\text{cur} \rightarrow \text{pre}[i] = \max(\text{cur} \rightarrow \text{left} \rightarrow \text{pre}[i], \text{cur} \rightarrow \text{right} \rightarrow \text{pre}[i] + \text{cur} \rightarrow \text{left} \rightarrow \text{sum});$
- $\text{cur} \rightarrow \text{suf}[i] = \max(\text{cur} \rightarrow \text{left} \rightarrow \text{suf}[i] + \text{cur} \rightarrow \text{right} \rightarrow \text{sum}, \text{cur} \rightarrow \text{right} \rightarrow \text{suf}[i]);$
- $\text{cur} \rightarrow \text{mid}[i] = \max(\text{cur} \rightarrow \text{left} \rightarrow \text{mid}[i], \text{cur} \rightarrow \text{right} \rightarrow \text{mid}[i], \text{cur} \rightarrow \text{left} \rightarrow \text{suf}[i] + \text{cur} \rightarrow \text{right} \rightarrow \text{pre}[i]);$
- 这个可以类比单点修改区间最大子段和理解

# Solution2

- 我们来看看这个需要支持什么操作
- 需要支持:
  1. 把一个凸函数加上一个常数
  2. 求一个凸函数 $a$ , 满足 $a[i] = \max( b[i], c[i] )$ , 其中 $b, c$ 为凸函数
  3. 求一个凸函数 $a$ , 满足 $a[i] = b[i] + c[i]$ , 其中 $b, c$ 为凸函数

# Solution2

- 可以发现这三个性质都满足
- 而且可以通过归并两个儿子的凸函数来得到cur的凸函数
- 于是预处理复杂度 $O(n \log n)$
- 每次查询的时候直接在线段树上查询区间即可
- 总复杂度 $O((n+m) \log n)$

# Luogu2824 [HEOI2016]排序

- 给出一个1到n的全排列，现在对这个全排列序列进行m次局部排序，排序分为两种：
- 1:(0,l,r)表示将区间[l,r]的数字升序排序
- 2:(1,l,r)表示将区间[l,r]的数字降序排序
- 最后询问第q位置上的数字。
- （其实可以加一个操作3:(2,x)表示查询x位置上的数字，还可以查区间和）



# Problem

- 区间排序好像不是很好维护？

# Solution1

- 既然只是最后要求查询一个位置的值，我们可以考虑二分答案
- 设二分的答案为ans
- 则把所有小于ans的设为0，大于ans的设为1
- 区间升序排序就是把所有0移到左边，1移到右边
- 区间降序排序就是把所有1移到左边，0移到右边

# Solution1

- 最后看查询的位置上是0还是1
- 就知道该往那边二分了~
- 总复杂度 $O(n \log^2 n)$

# Solution2

- 这个区间排序有类似于颜色段均摊的性质
- 考虑使用平衡树维护每个被排序的段，每个段内开一个数据结构维护段内排序后的信息
- 每次区间排序就是把一些段合并起来，并且把两边的段进行分裂
- 这里的数据结构我们就选择线段树合并
- 可以发现，对于每个被排序后的段，里面的值都是有序的，所以只需要值的信息

# Solution2

- 考虑区间排序
- 使用数据结构来维护每个段的信息
- 合并的时候复杂度看起来很对
- 问题在于分裂，每次分裂会将线段树里前k小的和其余节点分开

# Solution2

- 这里考虑“线段树合并”的势能
- 线段树合并为什么复杂度是对的？因为可以证明每次花费常数代价一定会减少一个节点
- 每次分裂只会增加 $O(\log n)$ 的势能
- 因为线段树每层只会访问到一个节点，最多将这—个节点变成两个，所以只会增加 $O(\log n)$ 个节点

# Solution2

- 于是我们外层使用一个set或者平衡树来维护
  - 每次查询现在在外层定位到内层的一棵线段树
  - 然后就可以查询对应的值了
- 
- 总时间复杂度 $O((n+m)\log n)$ , 空间复杂度 $O((n+m)\log n)$

# LOJ6189. Philosopher

给出一个长度为  $N$  的序列  $A$ ，要求维护以下两种操作：

- 1  $l$   $r$   $f$ : 将区间  $[l, r]$  内的元素按照升序 ( $f = 1$ ) 或降序 ( $f = 0$ ) 排序
- 2  $l$   $r$ : 询问  $[l, r]$  内的元素的积的十进制的最高位

对于所有数据,  $N, M \leq 200000, 1 \leq A_i \leq N$ 。



# Solution

# Luogu5612 [Ynoi2013]Ynoi

- 1.区间异或 $x$
  - 2.区间排序
  - 3.区间异或和
- 
- 空间32MB

# Solution

- 这道题由于是多次询问所以不能用前面题的做法了

# Solution

- 这个区间排序有类似于颜色段均摊的性质
- 考虑使用平衡树维护每个被排序的段，每个段内开一个数据结构维护段内排序后的信息
- 每次区间排序就是把一些段合并起来，并且把两边的段进行分裂

# Solution

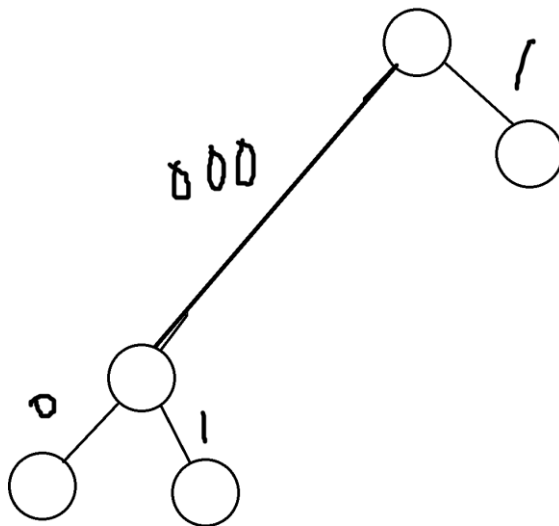
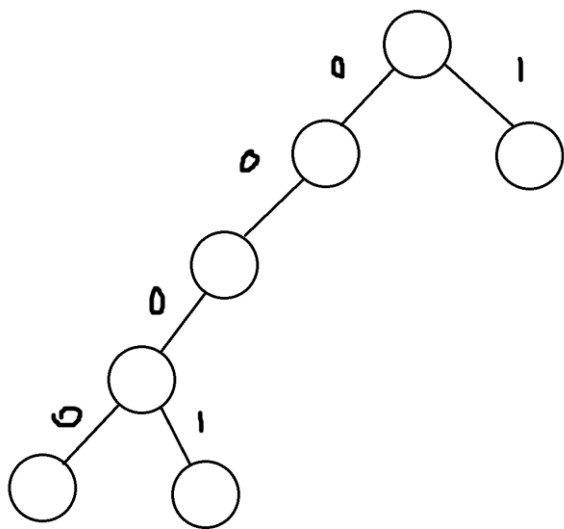
- 先只考虑区间排序
- 使用trie来维护每个段的信息
- 这里考虑“线段树合并”的势能，每次分裂只会增加 $O(w)$ 的势能，所以复杂度是对的

# Solution

- 区间xor可以在外层平衡树上打标记来实现
- 需要处理外层平衡树上的标记传到内层trie上的影响
- Trie上可以同时维护排序标记和xor标记来实现，同时需要维护trie的区间xor和
- 总时间复杂度 $O((n+m)\log n)$ ，空间复杂度 $O((n+m)\log n)$

# Technology

- 压缩trie:
- 用类似后缀树的思想, 把每个度数为2的点缩掉



- 总时间复杂度 $O((n+m)\log n)$ , 空间复杂度 $O(n)$

Thanks for listening