树套树

—— 成都七中　nzhtl1477

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 ai,bi,ci… 等不同的属性
- 每次对 l1<=ai<=r1,l2<=bi<=r2… 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.R-Tree

Range Tree

- 其实就是狭义上的树套树

树套树

- 能在 O(logn^d) 的复杂度内进行一次 d 维偏序的范围查询
- 能在 O(logn^d) 的复杂度内进行一次 d 维偏序的单点修改
- 空间为 O(nlogn^(d-1))，可以优化到 O(n(logn/loglogn)^(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(logv^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(logv^d)，空间复杂度 O(nlogv^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(logn)
- 平衡树每次访问是 O(log size) 的，所以树状数组套平衡树的最坏时间复杂度是 O(log^2n) 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 , [redacted] , * 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 <= 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(logn) 个节点的和
- 每次查询时用 O(logn) 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(logn) 个节点的平衡树
- 每次查询时用 O(logn) 个不相交节点中 <=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(logn^2)，空间复杂度 O(nlogn)

- 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(lognlogv)，空间复杂度 O(nlognlogv)

- 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(nlogn) 的空间复杂度）

线段树套树

- 套平衡树：时间复杂度 O(logn^2)，空间复杂度 O(nlogn)
- 套 Trie：时间复杂度 O(lognlogv)，空间复杂度 O(nlognlogv)

- 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(logn^2)，空间复杂度 O(nlogn)
- 套 Trie：时间复杂度 O(lognlogv)，空间复杂度 O(nlognlogv)

- 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[a]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<=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(logn^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(logn^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(logn^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((logn/loglogn)^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…）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(logn^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)+logn)，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(logn^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(logn^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(logn) 单次
- 空间复杂度 O(nlogn) 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

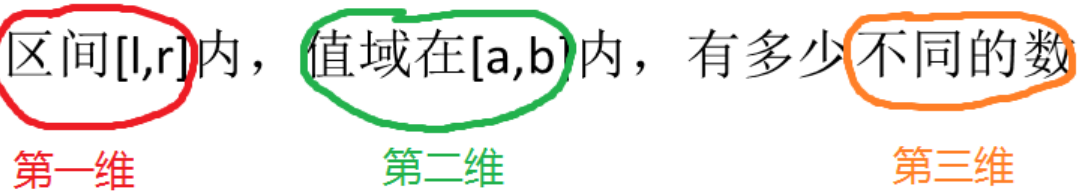
- 查询区间 [l,r] 内，值域在 [a,b] 内，有多少不同的数
- 强制在线， n,m<=1e6 ， 1024MB ， 8s

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



- 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(logn^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”

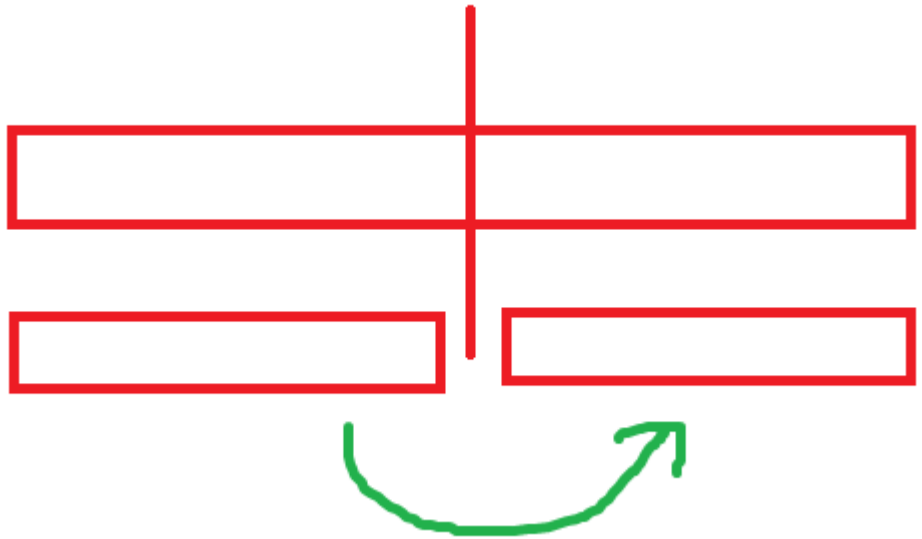
- 考虑如何用分治方法解决三维偏序

“CDQ 分治”

- 先考虑二维偏序也就是 $a_j \le a_i, b_j \le b_i$ 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 $b_j \le b_i$ 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”

- 现在我们有一个序列，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 1log 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 bj≤bi，说明 j 可以对后面加入的满足 cj≤ci 的 i 产生贡献了，把 cj 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 ci 的前缀和。
- 总时间复杂度 O(nlog^2n)

“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)+0(n)=0(nlogn)
- 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<=300，值 <=100,q<=2e5

Solution

- 翻译：有 n*m 个点，每个点有三个权值 xi ， yi ， zi
- 1. 把一个点的 zi 进行修改
- 2. 查询有多少点满足
- l1<=xi<=r1
- l2<=yi<=r2
- l3=zi

Solution

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

Solution

- 可以发现 xi， yi 只有 300 ， zi 只有 100，所以可以用三维树状数组维护
- 然后发现 zi 这一维只查询 [zi==l3] 的所有点
- 所以可以通过开 100 个二维树状数组来去掉 zi 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(logn^2) 单次
- (x,y,a) -> (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 ai,bi
- 每次修改一个点的 bi ，维护有多少点对 (i,j) 满足 [ai<aj&& bi>bj]

Solution

- 考虑每次修改的时候，对答案产生的贡献
- 假设原来是 (ai,bi) 变为 (ai,bi’)
- 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 (ai,bi) 有贡献的，加上所有和 (ai,bi’) 有贡献的点对个数

Solution

- 减去：第一维 >ai ，第二维 >bi 的，第一维 <ai ，第二维 <bi 的
- 加上：第一维 >ai ，第二维 >bi’ 的，第一维 <ai ，第二维 <bi’ 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^2n)

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 ai,bi
- 1. 加入 (l1,x),(l1+1,x)…(r1,x) 这些点
- 2. 求满足 l1<=ai<=r1 的所有点中 bi 的 kth

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)…(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

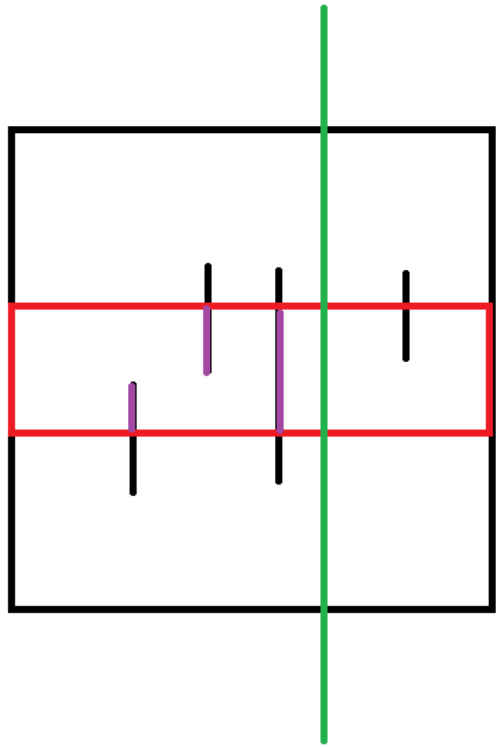
Solution

- 现在有了一个时间复杂度 O(logn^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(logn^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

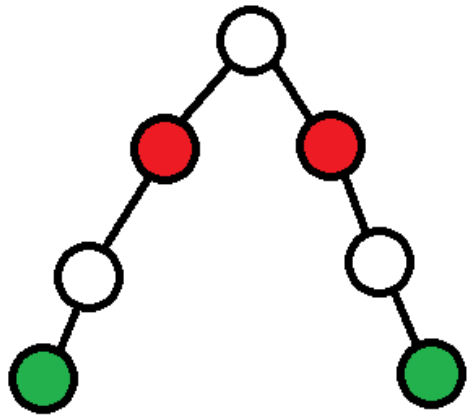
- 其实外部维护区间也是可以的，因为偏序问题基本上都是对称的，按什么顺序维护都是可以在同样的复杂度维护的
- 但是这个涉及到一个好不好写的问题…所以如果发现直接做比较麻烦可以试着换换偏序维护的顺序

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，求包含这个路径的路径里面权值 kth 是多少?
- 如果是查询这个路径包含的路径里面权值 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 <= ly <= ry <= 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 <=lu<=rx,ly<=lv<=ry 的所有点
- （当然还有 x 是 y 祖先的这种情况，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 ai 和 bi 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 kth

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

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^2n) 维护
- 总复杂度 O((n+m)log^2n)

- 注意常数…
- 小心细节…

Bzoj 3489

- 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 。
- 强制在线
- n<=1e5,m<=1e6

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 pi，后继为 qi，把询问看成点 (L,R)，有贡献的 i 满足 L∈(pi,i] 且 R∈[i,qi)，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。

- O(nlog^2n + mlogn)

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，……，$b-1$ 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 $1,2,\ldots,q$ 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] x [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] x [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 t1 时连通， t2 时断开
- 则其被加了 (m-t1)-(m-t2)=t2-t1 ，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(mlog^2n)

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∈ [1,l-1],b∈[l,r] 时，有 1-p 的概率不影响
- a∈[l,r],b∈[l,r] 时，有 1-2p 的概率不影响
- a∈[l,r],b∈[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(mlog^2n)

Summarization

- 省选里面的偏序类题一般都很裸，稍微推一推就可以转换成矩形操作问题
- 有几个常见的转换：
- 链 -> 两端的 DFS 序构成的矩形
- x 子树中距离 x<=k 的点 -> DFS 序一维，深度一维所构成的矩形
- 区间中不同数个数 -> 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…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(msqrt(n) + nsqrt(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
- `cur -> pre[i] = max( cur -> left -> pre[i] , cur -> right -> pre[i] + cur -> left -> sum );`
- `cur -> suf[i] = max( cur -> left -> suf[i] + cur -> right -> sum , cur -> right -> suf[i] );`
- `cur -> mid[i] = max( cur -> left -> mid[i] , cur -> right -> mid[i] , cur -> left -> suf[i] + cur -> right -> 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(nlogn)
- 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)logn)

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）

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(nlog^2n)

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

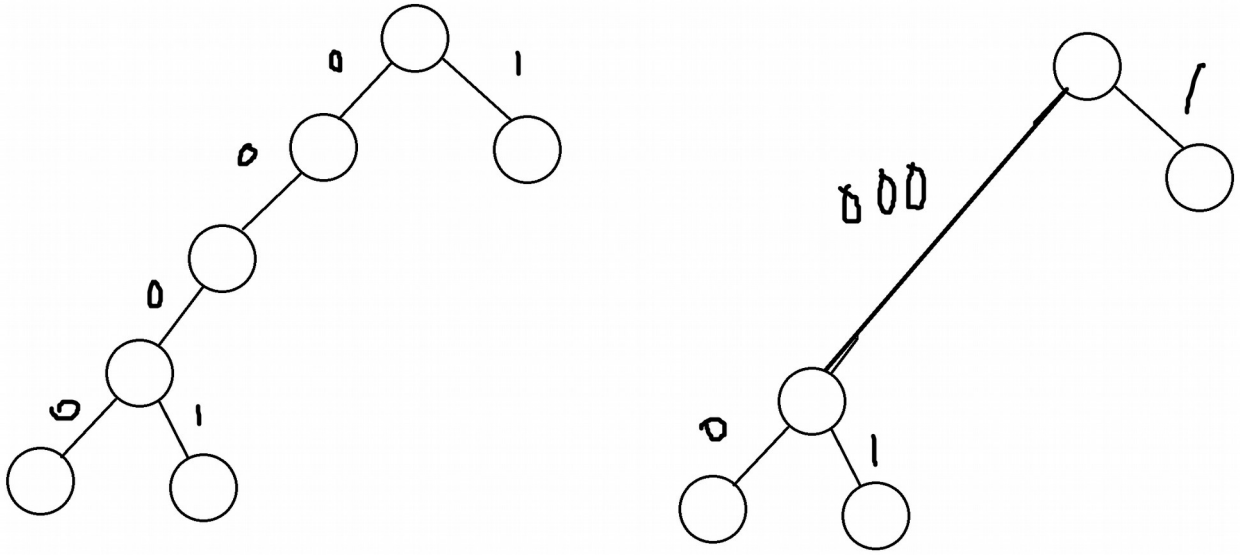
Solution

- 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)logn)，空间复杂度 O((n+m)logn)

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)logn)，空间复杂度 O(n)

Thanks for listening