数据结构题常见处理技巧

一些事实

- 常用的数据结构，通常是在值域上进行维护
- 今天涉及到的处理技巧，更多是在时间上进行考虑
- 众所周知，对于某个询问而言，只有时间更靠前的修改会有贡献
- 离线时可以不按照顺序进行回答
- 例如时间倒流，可以互换插入删除、分裂合并

CDQ分治

逆序对

- 给定n个数的数组：$a_1, a_2, \dots, a_n$，求逆序对个数
- 即统计$i < j \,\&\&\, a_i > a_j$的(i, j)对数
- 对每个j，统计$a_i > a_j (i \in [1, j-1])$的个数
- 一个a_i需要在a_j之前才可能产生贡献
- $Solve(l, r)$表示求下标在$[l, r]$内的(i, j)产生的贡献
- 将区间分成左右两部分$[l, mid]$和$[mid+1, r]$
- 分析两部分的贡献构成

逆序对

- 发现右边区间的i不可能对左边区间的j造成贡献
- 因此左半区间的贡献可以由$Solve(l, mid)$求出
- 右边区间的贡献可以分成两部分
- 第一部分$i, j \in [mid + 1, r]$
- $Solve(mid + 1, r)$
- 第二部分$i \in [l, mid], j \in [mid + 1, r]$
- 天然满足$i < j$的限制

逆序对

- 对于$i \in [l, mid], j \in [mid+1, r]$的贡献，在左右数字都有序的情况下，可以$O(n)$求出
- 并且可以$O(n)$将两部分归并，使得$[l, r]$区间内的数字有序，返回上一层继续求解
- 时间复杂度?
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
- $O(n\mathrm{log}n)$
- 这明明是归并排序，和CDQ分治有什么关系?

CDQ分治

- 操作序列：(C,Q,Q,C,C,Q,C,Q)，C对Q的贡献独立
- 前面的C会对后面的Q造成贡献
- 同样$solve(l,r)$表示计算下标在$[l,r]$内操作的贡献
- 将序列一分为二: (C,Q,Q,C），（C,Q,C,Q)
- 左半部分是子问题，递归解决
- 右半部分可以通过$solve(mid+1,r)$和(C,..,..,C)对(.,Q,.Q)的贡献算出

动态 → 静态

CDQ分治

- 一类离线动态问题，满足：
  - 静态版本能做，复杂度只和操作次数相关
  - C对Q的贡献独立
- 总操作个数为n，求时间复杂度?
- 提示：放到递归树上进行分析
- $O(logn) \times$ 查询 $+$ 总大小$O(nlogn)$的插入

陌上花开

- n朵花(a_i, b_i, c_i)，如果$a_i \geq a_j \&\& b_i \geq b_j \&\& c_i \geq c_j$，则称$i$比$j$更美丽，对每朵花求出它比多少花美丽$(n \leq 10^5)$
- 首先将完全一样的花合并，然后再做
- CDQ不是用来解决动态问题吗？和这题有什么关系？
- 将所有花进行三关键字的排序，把a看作时间，静态=>动态
- 再使用CDQ，动态=>静态

陌上花开

- $Solve(l,r)$表示计算下标在$[l,r]$内的花互相造成的贡献
- 左半部分是子问题，$Solve(l,mid)$
- 右半部分的贡献分为两部分
- 一部分是两朵花均属于右半部分，$Solve(mid+1,r)$可以求出
- 另一部分是左边的花对右边的花造成的贡献，静态问题
- 以 (b,c)为关键字对左右分别进行排序，然后树状数组维护
- 时间复杂度?
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n\log n) = O(n\log^2 n)$
- 注意每次清空数据结构

三维偏序

[HEOI2016/TJOI2016] 序列

- 序列$\{a_n\}$，m个形如(x,y)的修改表示将$a[x]$修改为y，找到一个最长子序列，使得该子序列在任意一种修改下都是不降的（也可以不修改），输出长度$(1 \le n,m \le 10^5)$
- 设$dp[x]$表示以下标x结尾的最长子序列长度
- $dp[x] = \max\{dp[y]\} + 1(y < x \&\& \operatorname{mx}[y] \le a[x] \&\& a[y] \le mi[x])$
- 和我们前面讲的有什么关系？
- 求dp值可以同时看作是询问和修改
- 需要先求出左半对右半的贡献，再递归右半部分
- 在递归树上中序遍历！

[HEOI2016/TJOI2016] 序列

- $Solve(l,r)$表示计算出$dp[l\sim r]$
- 左半部分$Solve(l,mid)$之后，需要先计算出左对右的贡献，再$Solve(mid+1,r)$
- 时间复杂度?
- $T(n)=2T\left(\frac{n}{2}\right)+O(n\log n)$
- $O(n\log^2 n)$

[CEOI2017] Building Bridges

- 长度为n的序列$\{h_n\}$和$\{w_n\}$，选出若干个下标$\{p_1, p_2, \dots, p_k\}$，p递增且$p_1 = 1, p_k = n$，最小化$\sum\left(h_{p_{i-1}} - h_{p_i}\right)^2 + \sum_{i \notin p} w_i$
- $2 \le n \le 10^5, 0 \le h_i, |w_i| \le 10^6$
- 设f_i表示i为最后一个位置时的最小代价
- $f_i = \min\limits_{j<i}\{f_j + \left(h_i - h_j\right)^2 + sw_{i-1} - sw_j\}$
- 设$X(j) = h_j, Y(j) = f_j + h_j^2 - sw_j$
- 对于两个决策点$X(j_1) < X(j_2)$且j_2更优，有$\frac{Y(j_2) - Y(j_1)}{X(j_2) - X(j_1)} \le 2h_i$

[CEOI2017] Building Bridges

- 最优的决策点一定在下凸壳上面
- 但是横坐标和斜率都不单调…
- 平衡树维护凸壳，查询时在凸壳上二分即可…
- 对于斜率优化，我们喜欢横坐标和斜率都单调
- 通过CDQ分治强行具有单调性
- 在计算左半对右半的贡献时，单调队列维护即可
- 如果每次分治都在两边暴力排序，时间复杂度为$O(nlog^2n)$

[CEOI2017] Building Bridges

- 观察到两边都是对h_i进行排序，可以看作二元组$(h_i, 0/1)$
- 在递归之前按照h_i排序一次，递归时重新计算0/1，每一层为线性
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\mathrm{log}n)$
- 注意因为是dp，所以仍然按照中序遍历的方式
- 通常优化1D/1D，CDQ论文中提供了两种做法，$O(nlog^2n)$或时空均为$O(nlogn)$
- 事先按照斜率排序，每一层计算$0/1$，归并时按照横坐标归并，能做到时间$O(nlogn)$，空间$O(n)$

CDQ分治小结

- 一类离线动态问题，满足：
  - 静态版本能做，复杂度只和操作次数相关
  - C对Q的贡献独立
- $O(logn) \times$ 查询 + 总大小O$(nlogn)$的构建
- 可以看作是在时间的线段树上递归，需要注意遍历顺序
  - dp需要严格按照时间顺序求出，中序遍历
  - 希望将数据结构归并，后序遍历
- 对于偏序问题，可以将一维排序后变成动态问题，再采用CDQ变为静态问题

整体二分

[国家集训队] 矩阵乘法

- $n \times n$的矩阵，q次询问，每次询问一个子矩阵的k小
- $1 \leq n \leq 500, 1 \leq q \leq 6 \times 10^4, 0 \leq a_{i,j} \leq 10^9$
- 可以发现答案具有可二分性
- 但是每次二分的mid不同，所需的状态构建的复杂度过高
- 二分的状态可以被不同的询问复用

整体二分

- $solve(l, r, S)$表示：答案区间$[l, r]$，答案属于该区间的询问集合S
- 二分答案mid，然后维护状态mid
- 对每个询问进行判定，并根据判定结果分为两个集合S_l和S_r
- 递归调用$solve(l, mid, S_l)$以及$solve(mid + 1, r, S_r)$
- 注意需要将S_r中询问减去左半部分的贡献
- 整个过程就是在答案的线段树上进行递归，每次将节点上的若干询问下放
- 时间复杂度?

整体二分

- 对于插入i，在答案的线段树上，共有$\log w$个区间包含它，因此会被插入$\log w$次
- 因此插入部分复杂度：$O(\log w) \times (插入 + 删除)$
- 对于询问i，在答案的线段树上，会一路下放$\log w$个区间，因此会询问$\log w$次
- 因此询问部分复杂度：$O(\log w) \times 询问$
- 对于这道题目，答案一定是出现过的数，$w = n$
- 总时间复杂度$O((n^2 + q)\log^2 n)$

[POI2011]MET-Meteors 流星

- 每次操作对区间加正数，回答每个位置达到a_i所需最小操作数
- 对操作时间进行二分
- 注意维护状态时不能使用前缀和，需要使用和操作、询问长度相关的复杂度
- 放这道题主要是有同学说觉得CDQ和整体二分很像…

[ZJOI2013] K大数查询

- 维护n个可重整数集，集合的编号为$1\sim n$，初始都为空集；有m个操作：1.在$[l,r]$内都插入一个数x；2.询问$[l,r]$的并集中，k大数；注意可重集的并不去除重复元素$\{1,1,4\} \cup \{5,1,4\} = \{1,1,4,5,1,4\}$
- $n, m \le 5 \times 10^4$
- 不仅询问可以下放，操作也可以下放
- 在二分过程中，操作能对询问造成影响：
  - 操作在询问之前
  - 操作在二分区间里

[ZJOI2013] K大数查询

- $Solve(l,r,S)$表示答案区间为$[l,r]$，操作序列为S
- 二分答案mid，并**按时间顺序**处理操作
- 如果当前操作为插入x：
- 若$x \le mid$，该操作并不处于二分区间，无需维护；并且对于将来右半部分的所有mid，该操作都不会有任何贡献；加入左区间
- 若$x > mid$，该操作处于二分区间，因此维护当前操作；被分到左区间的询问需要永久减去该询问的贡献；加入右区间
- 对于询问而言，如果分到左区间，需要永久减去分到右区间插入的贡献

[ZJOI2013] K大数查询

- 需要维护的东西为区间加区间和，因此线段树/树状数组维护即可
- 时间复杂度分析类似

整体二分小结

- 离线，具有答案单调性，可以二分回答询问，但是每次回答mid需要的状态不同，暴力维护状态代价过高
- 贡献需要满足交换、结合、可加减
- 整体二分构造了一棵以答案为下标的线段树，将操作从根节点开始下放，复用了mid的状态
- 时间复杂度$O(\log w) \times (插入 + 删除 + 询问)$

线段树分治

线段树分治

- 离线动态问题，有查询、插入、删除，没有删除操作我们会做...
- 考虑如何消除掉删除操作
- 对于每个插入，计算出他存在的时间区间，丢到每一刻的桶里
- 对每个时刻，将桶里的插入全部做一遍，得到对应时刻的状态
- 这样会直接乘一个$O(n)$...发现可以用线段树优化这个过程
- 将每个元素存在的时间拆分成log段，往线段树对应节点的桶丢
- 在遍历线段树时，儿子继承父亲的状态，回溯时撤销儿子的影响
- 时间复杂度$O(logT) \times (\text{插入} + \text{撤销}) + \text{查询}$
- 空间复杂度$O(logT) \times \text{操作}$

模板题

- n个点m条边的无向图，总时长为k，每条边有一个存在时间**区间**，问每个时刻整个图是否是二分图$(n, k \leq 10^5, m \leq 10^6)$
- 没有删除时可以用并查集做
- 利用线段树分治消除删除操作
- 儿子在回溯时需要撤销影响
- 用可撤回并查集维护
- $O(m\log k\log n)$

[FJOI2015] 火星商店问题

- n个商店，某个商店某个时刻会买入权值为v的商品（每个商店有一个特殊商品每个时刻都存在）。若干个顾客各自有一个参数 x ，询问在$[l,r]$商店购买的进货时间$[tl,tr]$的商品中，$v \textbf{ xor } x$的最大值。顾客和进货量总数为m。 $n,m,v,x \leq 10^5$
- max是可以拆成若干次的，询问可以拆成log段，然后在对应的线段树节点上查询
- 消除了时间的影响之后，发现只需要可持久化01Trie即可维护
- 时间复杂度$O(nlog^2n)$

二进制分组

二进制分组

- 动态问题，修改+询问，静态版本容易但不支持插入，而且强制在线
- 如果可以分批进行贡献，且询问的复杂度只和自身有关，可以尝试二进制分组

String Set Queries

- 维护一个小写字母字符串集合D，支持3种操作，共m次：
- 给出s，将s加入
- 给出s，将s删除
- 给出s，回答D中所有字符串在s中出现的次数和
- 强制在线
- $m \le 3 \times 10^5, \sum |s_i| \le 3 \times 10^5$

String Set Queries

- 首先删除可以看作是一次插入…
- 静态版本可以用AC自动机解决，但是AC自动机并不支持插入
- 考虑暴力重构
- 假设当前已经有了22个修改操作，$22 = 16 + 4 + 2$，我们将其分为3个AC自动机，上面分别是16,4,2个字符串，查询时在3个AC自动机上都做一遍，然后合并答案
- 考虑新加入一个串， $23 = 16 + 4 + 2 + 1$，此时我们将新加入的串单独开一个AC自动机
- 此时又加入一个串，$24 = 16 + 8$，此时我们将后3个AC自动机删掉，然后8个串暴力重构一个AC自动机

String Set Queries

- 就像玩2048一样…
- 时间复杂度?
- 对于查询，每次会查询O($\log m$)次，总复杂度为$O(\sum|s_i|\log m)$
- 对于插入，每个串只会被重构$O(\log m)$次，总时间复杂度为$O(26\sum|s_i|\log m)$