20zr省选线上集训day4

> 洪华敦

A. 【20省选集训day4】Manager

我们可以发现以下几点：

- 将 a_x 变成 10^5 后，只有 $(x, root)$ 上的点的年终奖会改变
- 假设一个点的子树大小为 k，则变化后年终奖只有第 $(k+1)/2$ 小和第 $(k+1)/2+1$ 小这两种可能性

具体的就是，我们设每个点 x 子树里第 $(size[x]+1)/2$ 小的值为 $m[x]$，第 $(size[x]+1)/2+1$ 小的值为 $p[x]$，那么如果它子树里的点 y 被修改了的话，若 $a_y \le m[x]$，则最后年终奖为 $p[x]$，即中位数往后移了一位，否则年终奖还是 $m[x]$

我们可以用主席树快速求出 $m[x]$ 和 $p[x]$，之后相当于对每个 y，统计它的所有满足 $a_y \le m[x]$ 的祖先的 $p[x]-m[x]$ 的和，这可以通过 dfs+树状数组计算

时间复杂度：$O(n \log n)$

B.【20省选集训day4】GCD再放送

考虑 $k=1$ 时的做法

我们可以统计对于每个 x，$gcd=x$ 的区间个数之和，设为 $f[x]$

而 $f[x]$ 不太好直接算，考虑计算 $g[x]=\sum_{x|d} f[d]$

假设有 m 个数满足他是 x 的倍数，那么这 m 个数构成的区间都应该贡献到 $g[x]$ 里

所以 $g[x]+=\sum_{i=1}^{m} C_m^i i!(n-i)!$

考虑 $k>0$，做法也类似

我们讨论一下区间的形式：

- 他是某个读入的序列的子区间，这个我们可以一开始对所有区间统计所有子区间 gcd 之和来计算，记得要乘 $n!$
- 他可以分成三部分：一个前缀和一个后缀以及若干个完整的序列

第一部分可以用前缀 gcd 只有 $O(\log n)$段的技巧去做，这里就不详细展开了

对于第二部分，我们计算 $g[x]$ 时，将序列分成以下两种：

- all 类序列：整个序列的 gcd 是 x 的倍数
- pre 类序列：序列有某些但不是全部前缀的 gcd 是 x 的倍数
- suf 类序列：序列有某些但不是全部后缀的 gcd 是 x 的倍数

可以发现一个序列可以同时为 pre 类和 suf 类

然后我们对区间进行分类讨论：

- 一个 suf 类序列的后缀+若干个 all 类序列+一个 all 类序列的前缀
- 一个 all 类序列的后缀+若干个 all 类序列+一个 pre 类序列的前缀
- 一个 all 类序列的后缀+若干个 all 类序列+一个 all 类序列的前缀
- 一个 suf 类序列的后缀+若干个 all 类序列+一个 pre 类序列的前缀

第一个和第二个类似，我们设所有 suf 类序列 一共有 $ssum$ 个后缀满足 gcd 是 x 的倍数，以及所有 all 类序列的长度之和为 $asum$，那么贡献就是：

$\sum_{i=0}^{m-1} C_{m-1}^{i} asum \times ssum \times i!(n-i-2)!$

相当于先枚举中间有 i 个 all 类序列，

然后枚举后面是哪个 all 类序列的前缀，再乘上这个序列的合法前缀数，这一步可以直接简化为 $asum$

然后乘上所有 suf 类序列的合法后缀和即可

对于第三个：

类似地维护一个 $psum$ 表示所有 pre 类序列的合法前缀和个数，然后式子和上面类似，只不过要乘的系数是 $psum \times ssum$

但注意，$psum \times ssum$ 后出现同一个序列被用两次的情况，因为一个序列可能同时是 pre 类和 suf 类的，这个需要减掉

对于第四个：

在维护长度之和的基础上，再维护长度平方之和就可以很简单地计算了

时间复杂度：$O(nd(n)\log n)$，但明显很难跑满

复杂度那么大主要问题在于上面计算贡献时要枚举 $i=1...m$，这一步其实可以用 fft 预处理，这样可以把复杂度降低到 $O(n\log^? n)$，但考虑优化不会太大所以没有加强到这个程度

C. 【20省选集训day4】dict

既然比较的是类似字典序一样的东西，考虑枚举 k 使得：

- A 和 B 在第 $p_{1...k}$ 小的元素上都是相同的
- A 的第 p_{k+1} 小的元素比 B 的第 p_{k+1} 小的元素要小

设 B 中第 p_i 小的是 b_i

那么相当于有一堆 (p_i,b_i)，其中 A 也包含这些 b_i，且 b_i 在 A_i 中的排名也是 p_i

考虑将 p_i 排序，相邻的两个 p 形成的区间：一个是排名区间，也就是 p 的差，一个是数值区间，也就是 b 的差，设这两个区间的长度为 PL,BL

那么相当于在这个区间中要给这 PL 个排名去选数，所以方案数是 C_{BL}^{PL}

所以我们就得到了一个 $O(n^3)$ 的做法了，枚举 k 和 A 的第 p_{k+1} 小的元素的值，然后对每个这样的区间去算个组合数乘起来

优化

要优化的话，可以发现当我们从 k 变成 $k+1$ 时，那些需要去计算的区间，只有某一个分成了两部分，其他的都不变，而我们枚举的 A 的第 p_{k+1} 小的值就是在这个要被分裂的区间里枚举的

所以我们可以用 set 维护这些区间，然后枚举 A 的第 p_{k+1} 小的值后，再用组合数去计算

于是得到了一个 $O(nm)$ 的算法

再优化

现在算法的主要瓶颈在于，需要枚举 A 的第 p_{k+1} 小的值，然后用组合数计算

假设 b_{k+1} 所在的权值区间为 $[BL, BR]$

那么 A 的第 p_{k+1} 小的值的枚举范围是 $[BL, b_{k+1})$

之后当 k 增加后，这个区间会分裂成 $[BL, b_{k+1})$ 和 $(b_{k+1}, BR]$

我们很自然地能想到启发式分裂，如果我们在枚举 A 的第 p_{k+1} 小值时能做到 $min(BR-b_{k+1}, b_{k+1}-BL)$ 的话，那么总的复杂度就是 $O(n\log n)$ 的

具体也很简单，本来是 $< b_{k+1}$，如果 $BR-b_{k+1}$ 较小，我们就枚举 $> b_{k+1}$ 的，然后总方案显然是一个很好算的组合数，于是我们的计算复杂度就降低到了 $min(BR-b_{k+1}, b_{k+1}-BL)$

于是总复杂度就是 $O(n\log n)$ 的