2017 Multi-University Training Contest 4 Solutions

Claris

2017 年 8 月 2 日

1 Big Integer

如果知道 $k-1$ 个数的个数 $c_1, c_2, ..., c_{k-1}$，那么方案数就是：

$$\frac{\left(\sum\limits_{i=1}^{k-1} c_i\right)!}{\prod\limits_{i=1}^{k-1} c_i!}$$

构造多项式 $f_i(x) = \sum\limits_{j=0}^{n} \frac{g_{i,j}}{j!} x^j$，则将这 $k-1$ 个多项式乘起来之后，每一项乘以 $i!$ 即可得到答案，可以用 NTT 在 $O(nk^2 \log(nk))$ 的时间内预处理出初始答案。

注意到我们只关心每个时刻答案的和，故可以在 DFT 意义下直接累加答案，最后再将结果 IDFT 回来。

对于单点修改操作，可以看成是给某个多项式叠加上一个只有一项系数不为 0 的多项式，观察 NTT 的公式：

$$y_n = \sum_{i=0}^{d-1} x_i \times \left(g^{\frac{P-1}{d}}\right)^{in} \bmod P$$

那么它在 DFT 后的结果是一个等比数列，故直接对原始多项式叠加一个等比数列即可完成修改。

对于叠加多项式，直接 $O(k)$ 重新计算乘积是不可取的，正确方法是对于每一项记录非 0 的乘积以及 0 的个数，单次修改时间复杂度为 $O(1)$。每次操作的时间复杂度为 $O(nk)$。

总时间复杂度 $O(nk^2 \log(nk) + mnk)$。

2 Classic Quotation

首先通过 KMP 算法预处理出 T 的 $next$ 数组，设：

$pref_i$ 表示 S 的前缀 i 与 T 进行 KMP 后 KMP 的指针到达了哪里。

$preg_i$ 表示 S 的前缀 i 中 T 出现的次数。

$suf_{i,j}$ 表示从 S 的后缀 i，从 j 指针开始 KMP，能匹配多少个 T。

那么前缀 i 拼接上后缀 j 中 T 的个数为 $preg_i + suf_{j,pref_i}$。

令 $preg$ 为 $preg$ 的前缀和，suf 为 suf 的后缀和，$s_{i,j}$ 表示 i 前面中 $pref$ 为 j 的个数，那么对于询问 L,R：

$$\begin{aligned} ans &= \sum_{i=1}^{L}\sum_{j=R}^{n} preg_i + suf_{j,pref_i} \\ &= (n-R+1)preg_L + \sum_{i=0}^{m-1} s_{L,i} \times suf_{R,i} \end{aligned}$$

以上所有数组均可以使用 KMP 和递推求出，时间复杂度 $O((n+k)m)$。

3 Counting Divisors

设 $n = p_1^{c_1}p_2^{c_2}...p_m^{c_m}$，则 $d(n^k) = (kc_1+1)(kc_2+1)...(kc_m+1)$。

枚举不超过 $\sqrt{r}$ 的所有质数 p，再枚举区间 $[l,r]$ 中所有 p 的倍数，将其分解质因数，最后剩下的部分就是超过 $\sqrt{r}$ 的质数，只可能是 0 个或 1 个。

时间复杂度 $O(\sqrt{r} + (r-l+1)\log\log(r-l+1))$。

4 Dirt Ratio

二分答案 mid，检验是否存在一个区间满足 $\frac{size(l,r)}{r-l+1} \leq mid$，也就是 $size(l,r) + mid \times l \leq mid \times (r+1)$。

从左往右枚举每个位置作为 r，当 r 变化为 $r+1$ 时，对 $size$ 的影响是一段区间加 1，线段树维护区间最小值即可。

时间复杂度 $O(n\log n\log w)$。

5 Lazy Running

取 $w = \min(d_{1,2}, d_{2,3})$，那么对于每一种方案，均可以通过往返跑 w 这条边使得距离增加 $2w$。也就是说，如果存在距离为 k 的方案，那么必然存在距离为 $k+2w$ 的方案。

设 $dis_{i,j}$ 表示从起点出发到达 i，距离模 $2w$ 为 j 时的最短路，那么根据 $dis_{2,j}$ 解不等式即可得到最优路线。

时间复杂度 $O(w\log w)$。

6 Logical Chain

对于每个询问，求出强连通分量，那么一个点数为 t 的强连通分量对答案的贡献为 $\frac{t(t-1)}{2}$。

利用 Kosaraju 算法，只需要在正反图各做一次 DFS 即可求出强连通分量。面对稠密图，Kosaraju 算法的瓶颈在于寻找与点 x 相连且未访问过的点。考虑用 bitset 来保存边表 g_x，以及未访问过的点集 S，那么只需要取出 $g_x \& S$ 内的所有 1 即可在 $O(\frac{n^2}{64})$ 的时间内求出强连通分量。

时间复杂度 $O(\frac{mn^2}{64})$。

7 Matching In Multiplication

首先如果一个点的度数为 1，那么它的匹配方案是固定的，继而我们可以去掉这一对点。通过拓扑我们可以不断去掉所有度数为 1 的点。

那么剩下的图中左右各有 m 个点，每个点度数都不小于 2，且左边每个点度数都是 2，而右侧总度数是 $2m$，因此右侧只能是每个点度数都是 2。这说明这个图每个连通块是个环，在环上间隔着取即可，一共两种方案。

时间复杂度 $O(n)$。

8 Phone Call

不难发现这个过程就是 Prim 算法求最小生成树的过程，用 Kruskal 算法同样正确。

将所有线路按代价从小到大排序，对于每条线路 (a,b,c,d)，首先把 a 到 b 路径上的点都合并到 LCA，再把 c 到 d 路径上的点都合并到 LCA，最后再把两个 LCA 合并即可。

设 f_i 表示 i 点往上深度最大的一个可能不是和 i 在同一个连通块的祖先，每次沿着 f 跳即可。用路径压缩的并查集维护这个 f 即可得到优秀的复杂度。

时间复杂度 $O(m \log m)$。

9 Questionnaire

取 $m=2$，必然存在一组可行解。

10 Security Check

设 $f_{i,j}$ 表示仅考虑 $a[1..i]$ 与 $b[1..j]$ 时，最少需要多少时间。

若 $|a_i - b_j| > k$，则 $f_{i,j} = f_{i-1,j-1} + 1$，否则 $f_{i,j} = \min(f_{i-1,j}, f_{i,j-1}) + 1$。

注意到后者只有 $O(nk)$ 个，可以暴力 DP，前者可以通过二分找到最大的 t，满足 i,j 往前 t 个均不冲突，然后再从某个后者状态转移过来。

时间复杂度 $O(nk \log n)$。

11 Time To Get Up

按题意模拟即可。

12 Wavel Sequence

设 $f_{i,j,k}$ 表示仅考虑 $a[1..i]$ 与 $b[1..j]$，选择的两个子序列结尾分别是 a_i 和 b_j，且上升下降状态是 k 时的方案数，则 $f_{i,j,k} = \sum f_{x,y,1-k}$，其中 $x < i, y < j$。暴力转移的时间复杂度为 $O(n^4)$，不能接受。

考虑将枚举决策点 x,y 的过程也 DP 掉。设 $g_{i,y,k}$ 表示从某个 $f_{x,y,k}$ 作为决策点出发，当前要更新的是 i 的方案数，$h_{i,j,k}$ 表示从某个 $f_{x,y,k}$ 作为决策点出发，已经经历了 g 的枚举，当

前要更新的是 j 的方案数。转移则是要么开始更新，要么将 i 或者 j 继续枚举到 $i+1$ 以及 $j+1$。因为每次只有一个变量在动，因此另一个变量恰好可以表示上一个位置的值，可以很方便地判断是否满足上升和下降。

时间复杂度 $O(n^2)$。

13 Yuno And Claris

先考虑如何求区间第 k 小值。对序列和权值都进行分块，设 $b_{i,j}$ 表示前 j 块中权值在 i 块内的数字个数，$c_{i,j}$ 表示前 j 块中数字 i 的出现次数。那么对于一个询问 $[l,r]$，首先将零碎部分的贡献加入到临时数组 tb 和 tc 中，然后枚举答案位于哪一块，确定位于哪一块之后再暴力枚举答案即可在 $O(\sqrt{n})$ 的时间内求出区间第 k 小值。

接着考虑如何实现区间 $[l,r]$ 内 x 变成 y 的功能。显然对于零碎的两块，可以直接暴力重构整块。对于中间的每个整块，如果某一块不含 x，那么无视这一块；否则如果这一块不含 y，那么只需要将 x 映射成 y；否则这一块既有 x 又有 y，这意味着 x 与 y 之间发生了合并，不妨直接暴力重构整块。因为有 c 数组，我们可以在 $O(1)$ 的时间内知道某一块是否有某个数。

考虑什么情况下会发生重构，也就是一个块内发生了一次合并的时候。一开始长度为 n 的序列会提供 $O(n)$ 次合并的机会，而每次修改会对零碎的两块各提供一次机会，故总合并次数不超过 $O(n+m)$，因此当发生合并时直接重构并不会影响复杂度。

那么现在每块中的转换情况只可能是一条条互不相交的链，只需要记录每个初值转换后是什么，以及每个现值对应哪个初值即可。遇到查询的时候，我们需要知道零碎部分每个位置的值，不妨直接重构那两块，然后遍历一遍原数组 a 即可得到每个位置的值。

在修改的时候，还需要同步维护 b 和 c 数组，因为只涉及两个权值，因此暴力修改 j 这一维也是可以承受的。

总时间复杂度 $O((n+m)\sqrt{n})$。