《数位 DP》解题报告

杭州第二中学 孙恒喆

2024 年 10 月 8 日

目录

1 题目描述

1.1 题目大意

给定长度为 $n-1$ 的字符串 s，对于一个长度为 n 的非负整数序列 a，定义其生成序列 b 为：

- $b_1 = a_1$；
- 对于 $i > 1$：
  - 若 $s_{i-1} = \texttt{A}$，则 $b_i = b_{i-1}$ and a_i。
  - 若 $s_{i-1} = \texttt{O}$，则 $b_i = b_{i-1}$ or a_i。
  - 若 $s_{i-1} = \texttt{X}$，则 $b_i = b_{i-1}$ xor a_i。

给定非负整数 k。接下来 q 组询问，每次给定一个 m，求有多少长度为 n 的整数序列 c 满足：

- 对于 $1 \le i \le n$，满足 $0 \le c_i < 2^k$。
- 存在至少一个长度为 n 的整数序列 a 满足：
  - 对于 $1 \le i \le n$，满足 $0 \le a_i \le c_i$。
  - 对于 a 的生成序列 b，满足 $b_n = m$。

由于答案很大，你只需要输出答案对 2^{32} 取模的结果。

1.2 数据范围

本题使用捆绑测试，你只有通过一个子任务的所有测试点，才能获得这个子任务的分数。

子任务编号	$n \le$	$k \le$	$q \le$	特殊性质	分值
1	4	5	200	无	10
2	20	8	20	无	10
3	200	16	1	无	10
4	200	16	200	无	10
5	200	30	1	$\text{popcount}(m) \le 16$	10
6	1000	30	1000	s 不包含 A	10
7	50	30	50	无	10
8	1000	30	1	无	10
9	200	30	200	无	10
10	1000	30	1000	无	10

对于 100% 的数据，$2 \le n \le 1000$，$1 \le q \le 1000$，$1 \le k \le 30$，$0 \le m < 2^k$。

2 解题过程

2.1 关于 2^k 复杂度做法

首先肯定需要找到一组 c 和一个 m 的快速判定方法，存在以下两种角度进行考虑：

- 如果我们正着考虑，可以发现结论：对于一组 c，满足条件的 m 是一段前缀，即形如 $[0,x]$。考虑归纳，假设当前符合条件的是前缀 $[0,x]$，那么有：
  - 对于 And c_i 操作，新的 x' 即为 $\min(x,c_i)$。
  - 对于 Or 或 Xor c_i 操作，新的 x' 可以表示为 x or c_i or $(\text{highbit}(x \text{ and } c_i)-1)$，特殊记 $\text{highbit}(0)=1$。

  上述结论不难证明。所以其实可以发现 Or 和 Xor 其实是相同的操作。
  直接模拟该做法即可做到 $O(n4^k)$，使用 FWT 优化上述做法即可做到 $O(n2^k k)$ 计算出所有 m 的答案。

- 如果我们倒着考虑，维护当前的目标值 m，每次进行如下更新：
  - 对于 And c_i 操作，需要满足 $c_i \geq m$，此时 m 不进行更新。
  - 对于 Or 或 Xor c_i 操作，我们从高位到低位考虑每一位：
    - 对于这一位上 c_i 是 0，不进行更新。
    - 对于这一位上 c_i 是 1 且 m 是 1，将 m 这一位更新为 0。
    - 对于这一位上 c_i 是 1 且 m 是 0，将 m 所有更低的位都更新为 0 并停止更新。

  上述结论可以通过调整法证明。
  对于第一个位置，我们看做 And 操作即可，并赋予初始值为 2^k-1。
  直接模拟该做法即可做到单次询问 $O(n4^k)$，使用 FWT 优化上述做法即可做到单次询问 $O(n2^k k)$。
  记 $\text{popcount}(m)=p$。更精细的实现可以做到单次询问 $O(n2^p p)$。

2.2 更优的复杂度

上述做法我们没有利用模数 2^{32} 的性质，那我们尝试利用一下。为了方便，接下来把 Xor 都叫做 Or。

继续上述倒着考虑的做法，本质上是维护 m，对于 And 就直接给系数乘 (2^k-m)，对于 Or 其实是将 m 中的一些 1 改为 0，对于不同的改法有不同的系数。具体地我们枚举一个 p 要求 m 对应的位是 0，对于比 p 高的位且 m 是 1，我们可以将其改为 0 也可以不改。对于 p 及以下的位清 0，此时的系数是 2^p。也可以不存在 p，对于所有 1 都可以改为 0 或不改，此时系数为 1。注意最后对 m 没有任何的要求。所以实际上每个位都是在一个后缀是 1，剩下的前缀是 0。

让我们尝试拆开 And 的贡献，即拆成 2^k，和若干个 -2^p，表示 p 这个位在 m 中为 1。这样拆贡献的好处在于指数和需要小于 32，否则不会产生任何贡献。考虑第 i 个位置的 Or 操作，

我们先找到 $1 \sim i-1$ 位置中 And 操作钦定的最低位的 1，记作 p，不存在则令 $p=k$。我们发现对于这个位置上的 Or 操作，对于这个 Or 操作的数而言，在 p 以下的位是可以任意填的，此时系数为 2^p。对于 p 及以上的位我们先留着，不过对于 p 以上的位我们不能进行“将更低的位赋值为 0”的操作，因为 p 这个位被钦定为 1。

在从前往后扫的过程中，p 会不断减小。我们可以用一个轮廓线描述 p 的变化，纵向是位，横向是 n 个位置。我们已经计算了 And 的贡献，和轮廓线下方的位的 Or 的贡献（即一些 2 的幂）。现在还剩下在轮廓线上方的 Or 的贡献。考虑每个位在 m 中是 1 的位，考虑其在轮廓线上方的部分是一个后缀 $[x,n]$，找到这个后缀中第一次被钦定为 1 的位置 y，那么在 $[x,y]$ 中它就是一个后缀是 1，其余前缀（也可以没有）是 0。也就是我们要在 $[x,y]$ 中找到一个 Or，在这个操作中修改到了这个位使其变成了 0（或者不存在）。所以这里的贡献就是 $[x,y]$ 区间中的 Or 的个数 $+1$。

我们维护一些信息，从前往后扫即可统计贡献。由于我们要记录上述的 p 的轮廓线，以及对每个位要看是否是第一次出现，我们需要状压下已经出现过的位的信息。我们同时记录轮廓线信息和已经出现过的位的信息，注意轮廓线信息只需要记录每次 Or 的时候 p 的值，也就是记录每个位第一次出现在轮廓线以上是在第几次 Or。有效的状态并不会很多，我们先搜出所有状态建立转移，再扫一次 n 个操作维护当前处在的状态，查询的时候枚举所有状态计算其系数即可。

对于每个位 p，我们在状态中记录一个非负整数 a_p。如果 $a_p=0$，表示其还没有出现过；如果 $a_p=1$，表示其出现过但不在轮廓线上；对于 $a_p>1$，表示其出现过，且在轮廓线上的占领的 Or 的个数为 a_p-1。可以发现，对于每个 p，其会恰好贡献 a_p 个 2^p，所以这个状态的本质为 ≤ 32 的拆分数的和。由于对于 $p=0$ 有两种表示（即 $a_0=0$ 或 $a_0=1$），因此实际上是两倍拆分数。

因此可以得到状态数，经过一些优化可以得到转移数在 3×10^5 级别。因此可以通过。

3 参考资料

暂无。