《拼数》命题报告

杭州学军中学 姜迅驰

摘 要

本文介绍了作者给校内训练命制的一道题《拼数》。解题过程从一个简单的算法开始，不断观察性质，优化复杂度，最后得到了一个巧妙的解法。

1 题目描述

1.1 题目大意

定义 $f(i)$ 将 i 的二进制形式写下所构成的字符串，例如 $f(3) =$ “11”、$f(6) =$ “110”。

你需要恰当地选择一个 1 到 n 的排列 P，使得字符串 $S = f(P_1) + f(P_2) + \cdots + f(P_n)$ 的字典序尽可能小。

输出 S 的前k位中的 1 的数量。

由于一些原因，输入和输出采用二进制形式。

1.2 数据规模

对于所有数据，$1 \le n < 2^{2000}$，$1 \le k \le |S|$。

- 子任务 1（7分）：$1 \le n \le 8$；
- 子任务 2（16分）：$1 \le n \le 10^5$；
- 子任务 3（26分）：$1 \le n \le 2^{60}$；
- 子任务 4（23分）：$1 \le n \le 2^{300}$；
- 子任务 5（28分）：无特殊限制。

时间限制：1s

空间限制：512MB

1.3 输入格式

从标准输入读入数据。

第一行一个不存在前导零的 01 串，以二进制形式给出了题目中的 n。

第二行一个不存在前导零的 01 串，以二进制形式给出了题目中的 k。

1.4 输入格式

输出到标准输出中。

共一行，一个不存在前导零的 01 串，以二进制形式给出题目的答案。

1.5 样例输入

```
1011
10010
```

1.6 样例输出

```
1000
```

1.7 样例说明

根据题意，一种可能的排列为（以下所有数均以二进制形式给出）：

`1000,100,1001,10,1010,101,1011,110,1,11,111`

将这些二进制首尾相接得到字符串 S，容易发现答案为 8。

2 记号与约定

对于一个字符串S，定义$S[i]$为S的第i个字符。

定义$S[l...r]$为S的第l个字符到第r个字符所构成的字符串，特殊地，若$l > r$，则$S[l...r] = \emptyset$。

定义$|S|$为S的长度。

定义$h(S)$为将一个01字符串S从左到右写下后构成的二进制数。

定义$|f(n)| = N$。

定义$x \times S$为将一个字符串S重复x遍之后得到的字符串。

定义S^{∞}为将S无限重复之后得到的一个无限长的字符串。

定义字典序为：若$|X| < |Y|$且$X = Y[1 \ldots |X|]$，或存在一个$i \le |X|$满足$X[1 \ldots i-1] = Y[1 \ldots i-1]$且$X[i] < Y[i]$，则$X$的字典序小于$Y$。

3 初步分析

3.1 算法一

枚举所有的合法排列 P，求出其生成的字符串中字典序最小的。

复杂度 $O(n! \times n \log n)$。

期望得分：7分。

3.2 算法二

引理 1. 对于两个字符串 X 和 Y，若 $X^\infty = Y^\infty$，则 $X + Y = Y + X$。

证明. 找到X^∞的最小循环节s，则可得

$$X = \frac{|X|}{|s|} \times s,$$

$$Y = \frac{|Y|}{|s|} \times s.$$

则

$$X + Y = Y + X = \frac{|X| + |Y|}{|s|} \times s.$$

□

引理 2. 对于两个字符串X和Y，若选择一个$1 \le k \le |X| + |Y|$，满足

$$X^\infty[1 \ldots k-1] = Y^\infty[1 \ldots k-1],$$

则

$$(X + Y)[k] = X^\infty[k],$$

$$(Y + X)[k] = Y^\infty[k].$$

证明. 不妨设$|X| < |Y|$。

若$k \le |X|$，则

$$(X + Y)[k] = X[k] = (X^\infty)[k],$$

$$(Y + X)[k] = Y[k] = (Y^\infty)[k],$$

若$|X| < k \le |Y|$，则

$$(X+Y)[k] = Y[k-|X|] = (X^\infty)[k-|X|] = (X^\infty)[k],$$

$$(Y+X)[k] = Y[k] = (Y^\infty)[k],$$

若$|Y| < k$，则

$$(X+Y)[k] = Y[k-|X|] = (X^\infty)[k-|X|] = (X^\infty)[k],$$

$$(Y+X)[k] = X[k-|Y|] = (Y^\infty)[k-|Y|] = (Y^\infty)[k].$$

□

引理 3. 对于两个字符串 X 和 Y，若 $X^\infty < Y^\infty$，则 $X+Y < Y+X$。

证明. 设 X^∞ 和 Y^∞ 的第一个不同的位置为 k，由引理二可得，

$$(X+Y)[1\dots k] = (X^\infty)[1\dots k],$$

$$(Y+X)[1\dots k] = (Y^\infty)[1\dots k].$$

可得

$$(X+Y)[1\dots k] < (Y+X)[1\dots k],$$

即$X+Y < Y+X$。

□

定理 1. 给出 n 个字符串 $A_1, A_2, \dots, A_n$，所有使得 $A_{P_1} + A_{P_2} + \cdots + A_{P_n}$ 的字典序最小的排列 P 均满足以下条件：对于任意 $1 \le i < n$，$(A_{P_i})^\infty \le (A_{P_{i+1}})^\infty$。

证明. 若存在一个i满足$(A_{P_i})^\infty > (A_{P_{i+1}})^\infty$，根据引理三可得

$$A_{P_i} + A_{P_{i+1}} > A_{P_{i+1}} + A_{P_i}.$$

则交换P_i与P_{i+1}更优。

而对于一对$(A_{P_i})^\infty = (A_{P_{i+1}})^\infty$，根据引理一可得

$$A_{P_i} + A_{P_{i+1}} = A_{P_{i+1}} + A_{P_i}.$$

则交换$(A_{P_i})^\infty$和$(A_{P_{i+1}})^\infty$后并不会对得到的字符串造成影响，即$(A_{P_i})^\infty$相等的字符串以任意顺序排列答案均为最优。

□

对于$f(1), f(2), \dots, f(n)$，将它们以 $f(i)^\infty$ 从小到大排序，再将它们拼接起来即可得到字符串 S。

复杂度 $\mathrm{O}(n \log^2 n)$。

期望得分：23分。

4 解法分析

将 1 到 n 的所有数以 $f(i)^\infty$ 从小到大排序，且在 $f(i)^\infty$ 相等的情况按照 i 从小到大排序，这样可以唯一确定一个排列 P。

定义$Q = P^{-1}$，即一个排列满足$Q_{P_i} = i$。

定义 c 为满足

$$\sum_{i=1}^{Q_c-1} |f(P_i)| \le k,$$

$$\sum_{i=1}^{Q_c} |f(P_i)| > k$$

的唯一整数。可以发现 $f(c)$ 在$S[1\dots k]$中只会保留一个前缀，而 $f(P_1), f(P_2), \dots, f(P_{Q_c-1})$ 在 $S[1\dots k]$中完整出现。

参考算法分为两个部分，第一部分为获得 c 的值，第二部分为通过 c 的值得到求出答案。

由于第二部分较为简单，此处会优先介绍第二部分。

4.1 第二部分

4.1.1 预处理

引理 4. 若$i < j$，且$|f(i)| = |f(j)|$，则$Q_i < Q_j$。

证明. 若$i < j$、$|f(i)| = |f(j)|$，则$f(i)^\infty < f(j)^\infty$，即$i$的位置在$j$的位置之前。可得$Q_i < Q_j$。 □

由引理四可得，同一个长度的所有数在P中是从小到大排列的。

定义

$$g(i, j) = \max\{k \mid |f(k)| = i, Q_k < Q_j\},$$

即排在 j 之前的最靠后的长度为 i 的数，之后会给出求出 $g(1, c), g(2, c), \dots, g(N, c)$ 的算法。

定理 2. 对于两个字符串 X 和 Y，若 $X^\infty \ne Y^\infty$，设 X^∞ 与 Y^∞的第一个不同的位置为 k，则$k \le 2\max(|X|, |Y|)$。

证明. 以下会证明若X^∞ 与 Y^∞的前$2\max(|X|, |Y|)$位相等，则$X^\infty = Y^\infty$。

若$\gcd(|X|, |Y|) = 1$，则对于X^∞ 与 Y^∞ 的前$|Y|$位，有$X[(i-1) \bmod |X|+1] = Y[i]$；对于$X^\infty$ 与 Y^∞ 的第$|Y|+1$位到$2|Y|$位，有$X[(i+|Y|-1) \bmod |X|+1] = Y[i]$。可得$X[i] = X[(i+|Y|-1) \bmod |X|+1]$。由于$\gcd(|X|, |Y|) = 1$，可得$X$的所有字符均相等。由$X[(i-1) \bmod |X|+1] = Y[i]$可推出$Y$的所有字符同样全相等，可得$X^\infty = Y^\infty$。

若$\gcd(|X|,|Y|)=d$，将X和Y中所有$\bmod d$相等的位单独取出考虑，可以采用类似于上述的方式证明它们均相等。这样同样可以说明$X^\infty=Y^\infty$。

$\square$

定义$T=f(c)^\infty[1\dots 2N]$，则 $g(i,c)$ 可能为$h(T[1\dots i])$或$h(T[1\dots i])-1$。

将$T[1\dots i]^\infty[1\dots 2N]$或$f(h(T[1\dots i])-1)^\infty[1\dots 2N]$与$T$比较字典序后即可在$\mathrm{O}(N)$的复杂度内得出答案。

注意对于$i=N$，若求出的 $g(i,c)>n$，则需要将 $g(i,c)$ 赋值为 n。

求出所有$g(i,c)$的复杂度为$\mathrm{O}(N^2)$。

4.1.2 统计$f(c)$的贡献

$f(c)$被截取的前缀长度为

$$k-\sum_{i=1}^{N} i\times(g(i,c)-2^{i-1}+1).$$

使用高精度计算以上式子即可得到$f(c)$被截取的长度l，求出$f(c)[1\dots l]$中有几个 1 即可。

4.1.3 统计$f(P_1),f(P_2),\dots,f(P_{Q_c-1})$的贡献

对于每个 $1\le i\le N$，需要将所有满足条件的长为i的数，即$2^{i-1},2^{i-1}+1,\dots,g(i,c)$中 1 的个数统计入答案。

假设$g(i,c)=\overline{a_1a_2\dots a_i}$，枚举所有的 $2\le j\le i$，若$a_j=1$，统计

$$[\overline{a_1a_2\dots a_{j-1}0}\times 2^{i-j},\overline{a_1a_2\dots a_{j-1}1}\times 2^{i-j})$$

中的数的贡献。这样共统计了$[2^{i-1},g(i,c))$中的数的贡献。

对于

$$[\overline{a_1a_2\dots a_{j-1}0}\times 2^{i-j},\overline{a_1a_2\dots a_{j-1}1}\times 2^{i-j})$$

中所有数的贡献可以分成两部分进行处理：

对于前最高j位的贡献统计，由于这个区间中最高j位都是相同的，设最高j位共有x个0，则答案需要加上$x\times 2^{i-j}$。

对于剩下的$i-j$位的贡献统计，可以看作$[0,2^{i-j})$中1的个数。对于任意$1\le k\le i-j$，第k位上有1的数的个数为2^{i-j-1}。则答案需要加上$(i-j)\times 2^{i-j-1}$。

使用以上步骤进行一个i 的贡献统计需要$\mathrm{O}(N)$次将答案加上$y\times 2^i$，其中y是一个较小的正整数。这可以使用以二的幂次为进制数的高精度以近似$\mathrm{O}(N)$的复杂度实现。

由于需要对 $1\le i\le N$ 全部统计贡献，这部分可以在近似$\mathrm{O}(N^2)$的时间复杂度内解决。

以上过程给出了求出了 c 之后高效解决此题的方法。

4.2 第一部分

以下描述的算法仅会提供第一部分的解法，第二部分的解法已在上述过程中被解决。

4.2.1 算法三

枚举c的长度，之后二分c的大小。对于二分时的一个值mid，采用以上方法求出 $g(1, mid), g(2, mid), \ldots, g(N,$
并计算总长

$$len = \sum_{i=1}^{N} i \times (g(i, mid) - 2^{i-1} + 1).$$

通过比较len与k的大小来决定二分区间的取舍。最后若满足 $0 \le k - len < |f(c)|$，则说明此时枚举的长度正好为c的长度。

复杂度$O(N^4)$。

期望得分：49分。

4.2.2 算法四

引理 5. 对于任意$2^{N-2} \le i < 2^{N-1} - 1$，有$Q_{i+1} - Q_i \le 2N + 2$。

证明. 对于所有$1 \le j < N$，满足长为j、且排在i与i+1之间的数只可能有两个：$f(i)[1 \ldots j]$与$f(i+1)[1 \ldots j]$。

长为N且排在i与$i+1$之间的数只可能有四个，$f(i)$+“0”、$f(i)$+“1”、$f(i+1)$+“0”、$f(i+1)$+“1”。

则排在i与$i+1$之间的数只可能有$2(N-1)+4 = 2N+2$个。

□

引理 6. $n - Q_{2^{N-1}-1} \le 1$。

证明. 排在$2^{N-1}-1$后面的数必定只由1构成，且长度大于$N-1$。

若$n = 2^N - 1$，则$n - Q_{2^{N-1}-1} = 1$，否则$n - Q_{2^{N-1}-1} = 0$。

□

引理 7. 若$g(N-1, c)$存在，则$Q_c - Q_{g(N-1,c)} \le 2N + 2$。

证明. 根据引理五和引理六可得，所有长为$N-1$的数的间距不超过$2N+2$，且最后一个长为$N-1$的数之后至多只会有一个数。则对于任意一个数，若找到在其之前最大的长为$N-1$的数，这个数和它的距离不会超过$2N+2$。

□

引理 8. $Q_{2^{N-2}} = 1$。

证明. 容易发现，排在2^{N-2}之前的数只有2^{N-1}。

□

根据引理七和引理八的结论，可以发现若c之前（不包括c）没有长为$N-1$的数，则c只有可能是2^{N-1} 或 2^{N-2}。

否则，可以通过算法三的二分方法来求出$g(N-1,c)$。设$g(N-1,c)=x$，之后需要求出排在x之后的$2N+2$个数，同时维护$len=\sum_{i=1}^{N} i\times(g(i,x)-2^{i-1}+1)$。

有关找出排在x之后的$2N$+2个数，可以通过对于所有$1\le i\le N$，维护$g(i,x)$并将$f(g(i,x)+1)^{\infty}[1\dots 2N]$插入字典树中。每次操作可以在字典树中$\mathrm{O}(N)$找到字典序最小的数并将其视作下一个数，同时需要将$len$加上这个数的长度。如果一次操作结束后发现$len>k$，则当前的数就是$c$。

这个部分的复杂度为$\mathrm{O}(N^2)$。

由于减少了枚举长度的步骤，二分的部分复杂度降为$\mathrm{O}(N^3)$。

期望得分：72分。

4.2.3 算法五

发现算法四的瓶颈在于二分，而二分之后的步骤的复杂度似乎不好优化。则可以尝试将二分改为逐位确定，维护一个数

$$x=\overline{b_1b_2\dots b_{N-1}}$$

和

$$len=\sum_{i=1}^{N} i\times(g(i,x)-2^{i-1}+1).$$

初始时

$$b_1=b_2=\cdots=b_{N-1}=0.$$

之后以$b_1,b_2,\dots,b_{N-1}$的顺序从高位到低位枚举，每次尝试将将b_i设为1。如果设为1之后发现$len>k$，则将b_i重新改为0。最后得到的x即为$g(N-1,c)$。

此处就需要支持修改x的一位，并快速维护len的值。

对于$g(N,x)$的贡献，可以使用$\mathrm{O}(N)$的时间单独处理。

对于$1\le i<N$，$g(i,x)=h(f(x)[1\dots i])$ 或 $h(f(x)[1\dots i])-1$。定义数列

$$d_1,d_2,\dots,d_{N-1}(0\le d_i\le 1),$$

满足

$$g(i,x)=h(f(x)[1\dots i])-d_i.$$

那么每次修改会造成的影响为：$h(f(x)[1\dots i])$可能被修改了一位、d_i被修改了。而这个修改对应到len上即为len变动了$i\times(y\times 2^j+z)$（$-1\le y,z\le 1$）。

如果可以使用一些方法快速地维护的$h(f(x)[1\dots i])$与d_i的变动，那么对于所有的$1 \le i < N$，可以使用以二的幂次为进制数的高精度以近似O(N)的复杂度将所有的$i \times (y \times 2^j + z)$累加。

对于$h(f(x)[1\dots i])$的修改，可以观察该次的修改操作是否影响到了x的前i位。如果是，执行对应的修改即可。

对于d_i的修改，定义

$$r_j = [f(x)^\infty[j] = f(x)^\infty[j+i]],$$

则若r_j为全1串则

$$d_i = [i = N - 1],$$

否则设r_j第一个为0的位为l，则

$$d_i = [f(x)^\infty[j] = f(x)^\infty[j+i]].$$

根据定理二，只需维护r的前$2(N-1)$项即可。而当修改x的一位时，对应到$f(x)^\infty$上只需修改2项，此时对应到r上只需修改4项。而之后问题就变成了修改r的一项，并快速给出r的第一个0的位置。

此处可以采用分块算法，由于在本题的数据范围下，$2(N-1) \le 4000$，此时将块大小设为64，并使用一个 unsigned long long 维护每块的信息。而块数同样不会超过64，可以使用 unsigned long long 维护每块是否存在0。在查询时，可以先使用位运算找到第一个存在0的块，并将在这个块中找到第一个0的位置。

这样就可以在近似O(1)的时间复杂度内维护单个d_i，而修改x的一位也达到了近似O(N)的复杂度。

在逐位确定的过程中需要进行O(N)次修改，这样便可以在近似O(N^2)的复杂度找到$g(N-1, c)$，并解决此题。

期望得分：100分。

5 总结

这道题最终的算法没有用的任何复杂的知识点，看上去是一道简单的高精度题，却考察了选手的思维能力、问题分析能力、代码能力。本题最终的算法将二分查找替换成逐位确定，并与本题的内容相结合，进行了优化，可以很好的锻炼选手思维的扩展性与创新性。本题的代码难度也较高，可以考验选手的代码基本功。

出题人希望使用该题启发大家，希望大家可以创造一些使用简单的知识点，却能够得到比较高的难度和比较优秀的区分度的好题。

参考文献

[1] 刘汝佳. 算法竞赛入门经典[M]. 清华大学出版社, 2009.

[2] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to algorithms[M]. MIT press, 2009.