《观虫我》命题报告

中国人民大学附属中学 叶李蹊

October 8, 2024

1 题目简述

给定一个长度为 2^n 的 0/1 序列 a，最初所有元素均为 0。需要执行 q 个操作：

1. **翻转操作**：给定一个下标 i，翻转 a_i 的值。
2. **查询操作**：给定一个下标 i，计算所有满足 $j \subseteq i$ 的 a_j 的异或和（即 j 的二进制位是 i 的子集）。

数据范围：$n = 32$，$q = 10^6$。

2 解题过程

2.1 算法一

2.1.1 算法 1.1

我们维护一个长度为 2^n 的 0/1 序列 a。

- 对于翻转操作，如果要翻转的下标是 i，我们枚举所有前 $n/2$ 位与 i 相同且后 $n/2$ 位是 i 的超集的下标 j，并翻转每个对应的 a_j。
- 对于查询操作，如果要查询的下标是 i，我们枚举所有后 $n/2$ 位与 i 相同且前 $n/2$ 位是 i 的子集的下标 j，并计算这些 a_j 的异或和。

时间复杂度：每个操作的时间开销为 $O(2^{n/2})$。

2.1.2 算法 1.2

考虑将 64 位压缩为一个整体。

我们将每 64 个相邻的序列元素存储在一个 `unsigned long long` 中，从而将序列长度缩小到原来的 1/64。

- 对于翻转操作，如果要翻转的下标是 i，我们对位置 $\lfloor i/64 \rfloor$ 的 `unsigned long long` 进行按位异或 $2^{i \bmod 64}$ 的操作。
- 对于查询操作，如果要查询的下标是 i，我们直接查询位置 $\lfloor i/64 \rfloor$ 的子集异或和（也是一个 `unsigned long long`），然后手动计算我们想要的位的异或和，它们对应着 $i \bmod 64$ 的子集。

时间复杂度：每个操作的时间开销为 $O\left(2^{(n-\log w)/2}\right)$，其中 w 为计算机的字长（本文中为 64）。

2.1.3 算法 1.3

我们代替前一半、后一半的划分，改为将 n 个二进制位划分为奇数位和偶数位。

这个变化提升了缓存友好性。虽然不改变时间复杂度，但显著减小了常数因子。

时间复杂度：每个操作的时间开销为 $O\left(2^{(n-\log w)/2}\right)$。

2.2 算法二

2.2.1 算法 2.1

算法一在翻转下标中 0 位占多数且查询下标中 1 位占多数时表现最差。
针对这样的数据，我们引入第三个做法：

- 对于翻转操作，如果要翻转的下标是 i，我们直接枚举 i 的所有子集 j，并翻转每个对应的 a_j。
- 对于查询操作，如果要查询的下标是 i，我们直接枚举 $\bar{i}$ 的所有子集 j（其中 $\bar{i}$ 表示 i 的按位取反），并计算这些 a_j 的异或和。

我们可以通过容斥原理来证明这个做法的正确性。
对于任意一对翻转和查询下标 (i,j)，考虑它们共同的中间下标 h 的个数（即 i 贡献给 h，且 h 贡献给 j 的下标）。
这些 h 可以是 i 中 1 位与 j 中 0 位交集的任意一个子集。

- 如果 $i \subseteq j$，则交集大小为 0，这样仅有 1 个 $h = \varnothing$，数目为奇数，贡献不被抵消。
- 如果 $i \not\subseteq j$，则交集大小至少为 1，这样恰有 2^k（$k \geq 1$）个 h，数目为偶数，贡献互相抵消。

采用这个做法，反而在翻转下标中 1 位占多数且查询下标中 0 位占多数时表现最差。
时间复杂度：每个操作的时间开销为 $O\big(2^{n-\log w}\big)$。

2.2.2 算法 2.2

使用不同做法处理每一位。
我们预定义一个初始设置 S，它是一个长度为 n 的字符串，每个字符为 A、B 或 C。
对于每一位：

- A：枚举查询中对应位为 1 的位，保持其他位不变。
- B：枚举翻转中对应位为 0 的位，保持其他位不变。
- C：枚举翻转中对应位为 1 的位和查询中对应位为 0 的位，将其他位变为 0。

共有 3^n 个可能的初始设置 S。每个初始设置对应不同的算法，这些算法都是正确的，但时间开销不同。我们的目标是选择一个好的初始设置，最小化总和时间开销。
对于给定的初始设置 S，设 A、B 和 C 分别为使用做法 A、B 和 C 的集合。使用初始设置 S 的算法如下：

- 对于翻转操作，如果要翻转的下标是 i，枚举所有在 $(\bar{i} \cap B) \cup (i \cap C)$ 中的位可以是任意值，并且 $i \cap (A \cup B)$ 中的位必须为 1 的下标 j。对每个这样的 j 执行翻转操作。
- 对于查询操作，如果要查询的下标是 i，枚举所有在 $(i \cap A) \cup (\bar{i} \cap C)$ 中的位可以是任意值，并且 $i \cap B$ 中的位必须为 1 的下标 j。计算这些 j 的 a_j 的异或和。

单个操作的时间开销为 2^k，其中 k 等于“可以是任意值”位的个数。
所有 q 个操作的总和时间开销是各个操作时间开销的总和。由于不同初始设置有不同的时间开销，我们希望选择一个能最小化总和开销的初始设置。
如果我们等概率随机选择一个初始设置，算法 2.3 中的分析表明这样做的期望时间开销为 $O\big((4/3)^n\big)$。
时间复杂度：期望每个操作的时间开销为 $O\big((4/3)^{n-\log w}\big)$。

2.2.3 算法 2.3

为了证明期望时间复杂度为 $(4/3)^n$，我们分析所有 3^n 个初始设置总和时间开销相加的和。
考虑一个翻转操作。对于所有 3^n 个初始设置，这个操作给所有初始设置贡献的时间开销之和等于 4^n，这与翻转下标 i 的具体值无关。

- 对于 i 中的每一个 0 位，如果初始设置为 B，则该位为一个“坏位”。
- 对于 i 中的每一个 1 位，如果初始设置为 C，则该位为一个“坏位”。

每个坏位都可以任意取 0/1，导致枚举空间扩大一倍。因此，总和时间开销的和等于 $(1+1+2)^n = 4^n$。

同理，对于一个查询操作，时间开销贡献之和也是 4^n。

然而，期望时间复杂度为 $(4/3)^n$ 只能保证对于任意 $c > 1$，存在 $1 - 1/c$ 的概率时间开销不超过 $c(4/3)^n$。这意味着存在较高的概率时间开销爆炸，这是不可接受的。

为缓解此问题，我们可以离线所有操作，随机选择 10 个初始设置，计算它们的实际时间开销，并选择其中开销最小的初始设置。这样，时间开销爆炸的概率降至 $1 - (1/c)^{10}$。例如，时间开销翻倍或更多的概率仅为 1/1024，这是可以接受的。

时间复杂度：期望每个操作的时间开销为 $O\big((4/3)^{n-\log w}\big)$。

2.3 附注

2.3.1 附注一

关于使用不同做法处理每一位的正确性证明。

我们可以这样推理：固定一对翻转和查询下标 (i, j)，考虑它们共同的中间下标 h 的个数是奇数还是偶数。对于每一位，考虑 h 可以取的值的个数：

- 如果做法为 A 或 B，且 i 在该位为 1、j 在该位为 0，则 h 在该位恰有 0 个可行取值。
- 如果做法为 C，且 i 在该位为 1、j 在该位为 0，则 h 在该位恰有 2 个可行取值。
- 其他情况，h 在该位恰有 1 个可行取值。

根据乘法原理，中间下标 h 的个数是每一位可行取值个数的乘积。这表明，如果至少有一位满足 i 在该位为 1 且 j 在该位为 0，那么 h 的数目为偶数（0 或者 2 的幂），贡献互相抵消。否则 h 的数目为 1，贡献不会抵消。也就是说，贡献仅在 $i \subseteq j$ 时发生。

2.3.2 附注二

关于为什么做法只有三个。

三个做法对应的是将一个矩阵分解为两个矩阵乘积的三个不同算式：

$$\text{(A)} \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} \mod 2$$

$$\text{(B)} \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} \mod 2$$

$$\text{(C)} \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} \mod 2$$

2.3.3 附注三

关于平均操作时间开销下界。

虽然 $(4/3)^n$ 是一个显然的上界，但我只能证明平均操作时间开销的下界为 $\left(\frac{\sqrt{13}-1}{2}\right)^n$，约为 1.303^n，对比朴素的折半算法为 1.414^n。

考虑这样生成输入数据：每个操作以 50% 的概率为翻转或查询。翻转下标 i 的每一位以概率 p 为 0，$1-p$ 为 1 独立随机。查询下标 i 的每一位以概率 q 为 1，$1-q$ 为 0 独立随机。

对于任意一个初始设置 S，记 A、B 和 C 分别表示使用做法 A、B 和 C 的集合，它们的占比为 a、b 和 c，其中 a, b, c 是 $1/n$ 的非负整数倍，并且满足 $a + b + c = 1$。对于这个初始设置：

- 翻转操作的平均时间开销为 $(1+p)^{bn}(2-p)^{cn}$。
- 查询操作的平均时间开销为 $(1+q)^{an}(2-q)^{cn}$。

一个恶意的输入数据会选择 (p, q) 来最大化底数：

$$\max\left((1+p)^b(2-p)^c, (1+q)^a(2-q)^c\right)$$

通过指定 $p = q = \frac{5-\sqrt{13}}{2} \approx 0.697$，也就是方程 $1 + x = (2-x)^2$ 的解，我们发现 $(2-x)^{2\max(a,b)+c} \geq 2 - x \approx 1.303$。当且仅当 $a = b$ 时取到等号。

因此，按照这样生成数据，无论初始设置如何，都无法实现小于 $\frac{\sqrt{13}-1}{2}$ 的底数。

值得说明的是，如果已经明确输入数据是按照上述方式生成的，则存在一个通用的三元组 (a,b,c)，无论参数 (p,q) 如何，都能实现底数 $\frac{\sqrt{13}-1}{2}$。该三元组为 $\left(\frac{1-x}{2},\frac{1-x}{2},x\right)$，其中 $x=\frac{1}{\sqrt{13}}\approx 0.277$，大约是 $(36\%,36\%,28\%)$。可以验证，当 $(a,b,c)=\left(\frac{1-x}{2},\frac{1-x}{2},x\right)$ 时，$(1+p)^{(1-x)/2}(2-p)^x$ 的最大值确实为 $\frac{\sqrt{13}-1}{2}$。

注意，在该三元组中，方法 C 的使用频率低于 1/3。少使用 C 大概是因为其在两种情况下会出问题（翻转的 1 位和查询的 0 位），而方法 A 和 B 仅在一种情况下会出问题。

尽管重要的是平均时间，但方差也不可忽视。单个操作的时间开销标准差有一个上界，称为常数的 n 次方，q 个操作平均时间开销的标准差则与 $\sqrt{q}$ 成反比。但对于 $q=10^6$ 来说，这个偏差其实不可忽视。

基于我能够生成的数据，即使当 $(a,b,c)=(1/3,1/3,1/3)$ 而非 $(36\%,36\%,28\%)$ 时，1.303^n 仍比 1.333^n 预测实际表现更加准确。然而，我没有能力证明这一点。

3 命题思路

这个问题本身是一个经典问题，算法 1.1 对于学习信息学奥林匹克竞赛的人耳熟能详。

三年前，我就思考过是否能将这个问题的时间复杂度优化到 $O(2^{n/2})$ 以下。

在 2024 年 10 月，我重新审视了这个问题，并询问了许庭强学长他是否知道这个问题已知的最优复杂度。

许庭强学长声称目前的最优复杂度是 $O(2^{n/3})$，但他忘记了具体的算法。

随后，我尝试自行还原该算法。有了复杂度的提示，经过一番思考，我推导出了如上所述的期望 $O\big((4/3)^n\big)$ 的算法。我与许庭强学长讨论后，发现他提到的算法实际上并不存在。

4 致谢

这个问题暂未找到参考资料。

特别感谢郭羽冲学长提供了一个随机化的 $O\left(\frac{2^{(n-\log w)/2}}{n^2}\right)$ 算法，书写了这个算法，并参与了验题的过程。

特别感谢许庭强学长给予我的灵感与启发。

特别感谢时庆钰学长提供的技术支持。

感谢葛致远、翟鹏昊、赵海鲲、陈诺、许淇文同学，王梓霁学长与我一起讨论此问题。

题目标题《观虫我》及题记摘自现代词作者慕清明（笔名）创作的一首歌曲。