蒙哥马利算法—一个梦幻般的的技术

FAP4算法通过计算错误的答案并且知道最后他能够在最后得出正确结果。

蒙哥马利发明的一个模乘算法甚至更奇怪：他计算答案，并加到一个完全不同的和上面来得到我们想要的结果。

让我们看看我们熟悉的例子，789098x123456，但是这次我们换做计算

789098x123456÷1000000。为了达到这个目的，我们将采用“移位

并相加“的技术，但是和从789098的左手边开始并且每次乘以

10不同，我们从右手边开始并且每次除以10：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 × 123456 = |  |  | 9 | 8 | 7 | 6 | 4 | 8 |  |  |  |  |  |  |  |
| ÷ 10 = |  |  |  | 9 | 8 | 7 | 6 | 4 | . | 8 |  |  |  |  |  |
| 9 × 123456 = | + | 1 | 1 | 1 | 1 | 1 | 0 | 4 |  |  |  |  |  |  |  |
| 9.8 × 123456 = |  | 1 | 2 | 0 | 9 | 8 | 6 | 8 | . | 8 |  |  |  |  |  |
| ÷ 10 = |  |  | 1 | 2 | 0 | 9 | 8 | 6 | . | 8 | 8 |  |  |  |  |
| 0 × 123456 = |  |  |  |  |  |  | + | 0 |  |  |  |  |  |  |  |
| 0.98 × 123456 = |  |  | 1 | 2 | 0 | 9 | 8 | 6 | . | 8 | 8 |  |  |  |  |
| ÷ 10 = |  |  |  | 1 | 2 | 0 | 9 | 8 | . | 6 | 8 | 8 |  |  |  |
| 9 × 123456 = | + | 1 | 1 | 1 | 1 | 1 | 0 | 4 |  |  |  |  |  |  |  |
| 9.098 × 123456 = |  | 1 | 1 | 2 | 3 | 2 | 0 | 2 | . | 6 | 8 | 8 |  |  |  |
| ÷ 10 = |  |  | 1 | 1 | 2 | 3 | 2 | 0 | . | 2 | 6 | 8 | 8 |  |  |
| 8 × 123456 = |  | + | 9 | 8 | 7 | 6 | 4 | 8 |  |  |  |  |  |  |  |
| 8.9098 × 123456 = |  | 1 | 0 | 9 | 9 | 9 | 6 | 8 | . | 2 | 6 | 8 | 8 |  |  |
| ÷ 10 = |  |  | 1 | 0 | 9 | 9 | 9 | 6 | . | 8 | 2 | 6 | 8 | 8 |  |
| 7 × 123456 = |  | + | 8 | 6 | 4 | 1 | 9 | 2 |  |  |  |  |  |  |  |
| 7.89098 × 123456 = |  |  | 9 | 7 | 4 | 1 | 8 | 8 | . | 8 | 2 | 6 | 8 | 8 |  |
| 0.789098 × 123456 = |  |  |  | **9** | **7** | **4** | **1** | **8** | **.** | **8** | **8** | **2** | **6** | **8** | **8** |

实际上，这正是倒过来计算原来的乘法的和，但有一点不同。当我们做加法的时候，进位总是从右到左的产生，并且在原来的方法里，这意味着直到我们完成整个计算之前我们不能确定结果的任何一个数字的值。但是现在我们正在从右手边产生结果然而这些进位仍然会流向左边，并且意味着数字一旦越过到小数点的右边，它的数字就不会改变。

**转换到模的形式**

现在假设我们想要计算789098 × 123456 ÷ 1000000 modulo 876543. 之前我没有提到过 模除，但是它的定义很简单：X ÷ Y (mod R) 就是满足这样条件的数字Z，它满足 Z × Y = X (mod R).

让我们再一次看看这个乘法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 × 123456 = |  |  | 9 | 8 | 7 | 6 | 4 | 8 |
| 现在我们想要除以10，但是我们不能这样做，因为这个数字不能被10整除。所以我们加上876543的一个倍数，之所以我们能这样做，是因为我们正在对876543取模。 | | | | | | | | |
|  | + | 3 | 5 | 0 | 6 | 1 | 7 | 2 |
| 8 × 123456 = | = | 4 | 4 | 9 | 3 | 8 | 2 | 0 |
| ÷ 10 = 0.8 × 123456 = |  |  | 4 | 4 | 9 | 3 | 8 | 2 |
| 9 × 123456 = | + | 1 | 1 | 1 | 1 | 1 | 0 | 4 |
| 9.8 × 123456 = | = | 1 | 5 | 6 | 0 | 4 | 8 | 6 |
| Another multiple of 876543 | + | 7 | 0 | 1 | 2 | 3 | 4 | 4 |
|  | = | 8 | 5 | 7 | 2 | 8 | 3 | 0 |
| ÷ 10 = 0.98 × 123456 = |  |  | 8 | 5 | 7 | 2 | 8 | 3 |
| 0 × 123456 = |  |  |  |  |  |  | + | 0 |
| 0.98 × 123456 = | = |  | 8 | 5 | 7 | 2 | 8 | 3 |
| Another multiple of 876543 | + | 7 | 8 | 8 | 8 | 8 | 8 | 7 |
|  | = | 8 | 7 | 4 | 6 | 1 | 7 | 0 |
| ÷ 10 = 0.098 × 123456 = |  |  | 8 | 7 | 4 | 6 | 1 | 7 |
| 9 × 123456 = | + | 1 | 1 | 1 | 1 | 1 | 0 | 4 |
| 9.098 × 123456 = | = | 1 | 9 | 8 | 5 | 7 | 2 | 1 |
| Another multiple of 876543 | + | 2 | 6 | 2 | 9 | 6 | 2 | 9 |
|  | = | 4 | 6 | 1 | 5 | 3 | 5 | 0 |
| ÷ 10 = 0.9098 × 123456 = |  |  | 4 | 6 | 1 | 5 | 3 | 5 |
| 8 × 123456 = |  | + | 9 | 8 | 7 | 6 | 4 | 8 |
| 8.9098 × 123456 = | = | 1 | 4 | 4 | 9 | 1 | 8 | 3 |
| Another multiple of 876543 | + | 7 | 8 | 8 | 8 | 8 | 8 | 7 |
|  | = | 9 | 3 | 3 | 8 | 0 | 7 | 0 |
| ÷ 10 = 0.89098 × 123456 = |  |  | 9 | 3 | 3 | 8 | 0 | 7 |
| 7 × 123456 = |  | + | 8 | 6 | 4 | 1 | 9 | 2 |
| 7.89098 × 123456 = | = | 1 | 7 | 9 | 7 | 9 | 9 | 9 |
| Another multiple of 876543 | + | 6 | 1 | 3 | 5 | 8 | 0 | 1 |
|  | = | 7 | 9 | 3 | 3 | 8 | 0 | 0 |
| ÷ 10 = **0.789098 × 123456 =** |  |  | **7** | **9** | **3** | **3** | **8** | **0** |

如果你喜欢可以检验这个结果，把它乘以1000000并且检查他是否等于770211模876543.

这个技术好的一点是对于如何决定模的倍数，仅仅取决于结果的最后一位不会被任何进位影响到的数字。所以 保存进位架构 在这里很合适。

**得到我们真正想要的答案**

你可能会看到，当你想要A × B 的时候，并不是想要 A × B ÷ 1000000. 先把疑问搁置一会，先看看如果我们把A和B都乘以一百万（1000000）会有什么事情发生：

1000000×A × 1000000×B ÷ 1000000 = 1000000 × A × B

换句话说，如果你把两个数都乘以1000000，蒙哥马利乘法会给你

一个他们乘积的1000000倍的结果。可以这样去乘两个数字：

1. 把他们两个每个都乘以1000000（模 R）.
2. 运用蒙哥马利乘法.
3. 把结果除以1000000（模R）

这看起来是没必要的详细说明，并且确实上如果你仅仅把蒙哥马利乘法当成模乘法的一种形式的时候它确实没有什么用处。但是当提到它的本质的时候，就是当你需要做一些列连续的模乘的时候。比如说，这里是如何计算y的25次幂。我将用“\*”代表“蒙哥马利 乘这些数字模R”。

Y = 1000000 × y (modulo R)

Y2 = Y \* Y

Y4 = Y2 \* Y2

Y6 = Y4 \* Y2

Y12 = Y6 \* Y6

Y24 = Y12 \* Y12

Y25 = Y24 \* Y

y25 = Y25 ÷ 1000000 (modulo R)

所有这些中间不够都像我们不得不用过的普通模乘算法一样，所以第一个和最后一个步骤就是所有技巧的步骤。

对于最后一步，我们能使用一个蒙哥马利乘法来实现除法：

y25 = Y25 \* 1

对于第一步，我们能用另外一个蒙哥马利乘法：

Y = y \* 1000000000000

它将自动的除以1000000并且这样我们得到了1000000× y，但是这里

有一点要注意，因为蒙哥马利乘法不能处理大于模的数字。所以我们

预计算（使用一些其他方法，也许甚至以软件形式）

TRILLION = 1000000000000(modulo R)

然后第一步我们能写成

Y = y \* TRILLION

所以使用蒙哥马利乘法的所有额外的开销加起来就是两个额外的乘法，

一个是在开始，一个在结束的时候，加上仅仅对特定的R值所针对TRILLION的预计算。

**二进制中更加简单**

十进制里很容易解释，但是蒙哥马利乘法在二进制更加容易解释。1000000的地位可以被一些2 的合适的幂取代，但是关键的简化的地方是加模的倍数的计算变得更加简单了。

规则就是：如果当前数字是奇数，在你二等分它之前加上R；如果它是偶数，直接二等分它。

**但是它有用么**？

蒙哥马利算法确实是一个优美并且出人意料的算法，而且我也嫉妒蒙哥马利能发现它。

如果大多数的工作对于加密芯片来说是进行模幂计算，或者模几乎不变（比如 在Diffiel-Hellman 密钥交换中）或者你能粗处预计算的TRILLION，那么他就非常有效率；但是总体来说它没有我的算法或者Brickell’s的好，因为他们都不需要预计算以及开头和结尾的乘法。