OT：n 位整数相乘问题

戴若石

南京大学计算机科学与技术系
171860004@smail.nju.edu.cn

2019 年 4 月 12 日

Contents

Contents

朴素算法

朴素算法

乘法的竖式计算

$B = \sum_{i=1}^{n} b_i \cdot 10^{i-1}$，则 $AB = \sum_{i=1}^{n} Ab_i \cdot 10^{i-1}$。

朴素算法

乘法的竖式计算

$B = \sum_{i=1}^{n} b_i \cdot 10^{i-1}$，则 $AB = \sum_{i=1}^{n} Ab_i \cdot 10^{i-1}$。

朴素算法

乘法的竖式计算

$B = \sum_{i=1}^{n} b_i \cdot 10^{i-1}$，则 $AB = \sum_{i=1}^{n} Ab_i \cdot 10^{i-1}$。

$$
\begin{array}{cccccc}
 & & & 1 & 2 & 3 \\
\times & & & 4 & 5 & 6 \\
\hline
 & & & 7 & 3 & 8 \\
 & & 6 & 1 & 5 & \\
 & 4 & 9 & 2 & & \\
\hline
 & 5 & 6 & 0 & 8 & 8
\end{array}
$$

朴素算法

乘法的竖式计算

$B = \sum_{i=1}^{n} b_i \cdot 10^{i-1}$，则 $AB = \sum_{i=1}^{n} Ab_i \cdot 10^{i-1}$。

$$
\begin{array}{cccccc}
 & & & 1 & 2 & 3 \\
\times & & & 4 & 5 & 6 \\
\hline
 & & & 7 & 3 & 8 \\
 & & 6 & 1 & 5 & \\
 & 4 & 9 & 2 & & \\
\hline
 & 5 & 6 & 0 & 8 & 8
\end{array}
$$

时间复杂度：$O(n^2)$

分治算法

未优化的分治算法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^{n} + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。

分治算法

未优化的分治算法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^{n} + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。

- 在计算两个 n 位整数相乘时，用到了 4 个 $\frac{n}{2}$ 位整数相乘,3 个 n 位整数加法以及 2 个移位操作。加法和移位操作的时间复杂度为 $O(n)$。

分治算法

未优化的分治算法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。

- 在计算两个 n 位整数相乘时，用到了 4 个 $\frac{n}{2}$ 位整数相乘,3 个 n 位整数加法以及 2 个移位操作。加法和移位操作的时间复杂度为 $O(n)$。
- 设时间复杂度为 $T(n)$，则有如下递推式：

$$T(n) = \begin{cases} O(1), n = 1 \\ 4\,T(\frac{n}{2}) + O(n), n > 1 \end{cases}$$

分治算法

未优化的分治算法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。

- 在计算两个 n 位整数相乘时，用到了 4 个 $\frac{n}{2}$ 位整数相乘,3 个 n 位整数加法以及 2 个移位操作。加法和移位操作的时间复杂度为 $O(n)$。
- 设时间复杂度为 $T(n)$，则有如下递推式：

$$T(n) = \begin{cases} O(1), n = 1 \\ 4\,T(\frac{n}{2}) + O(n), n > 1 \end{cases}$$

- 则 $T(n) = O(n^2)$。

Contents

Karatsuba 乘法

- 之前分治算法的复杂度递推式：$T(n) = 4\,T(\frac{n}{2}) + O(n)$。

Karatsuba 乘法

- 之前分治算法的复杂度递推式：$T(n) = 4\,T(\frac{n}{2}) + O(n)$。
- 想要降低时间复杂度，就要想办法减少乘法操作的次数。

Karatsuba 乘法

- 之前分治算法的复杂度递推式：$T(n) = 4\,T(\frac{n}{2}) + O(n)$。
- 想要降低时间复杂度，就要想办法减少乘法操作的次数。

Karatsuba 乘法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。
其中 $(ad + bc) = (a - b)(d - c) + ac + bd$，故
$AB = ac \cdot 10^n + ((a - b)(d - c) + ac + bd) \cdot 10^{\frac{n}{2}} + bd$。

Karatsuba 乘法

- 之前分治算法的复杂度递推式：$T(n) = 4\,T(\frac{n}{2}) + O(n)$。
- 想要降低时间复杂度，就要想办法减少乘法操作的次数。

Karatsuba 乘法

$A = a \cdot 10^{\frac{n}{2}} + b$
$B = c \cdot 10^{\frac{n}{2}} + d$
故 $AB = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$。
其中 $(ad + bc) = (a - b)(d - c) + ac + bd$，故
$AB = ac \cdot 10^n + ((a - b)(d - c) + ac + bd) \cdot 10^{\frac{n}{2}} + bd$。

现在只需要计算 $ac, bd, (a - b)(d - c)$ 这 3 个乘法了。

Karatsuba 乘法伪代码

Algorithm 1 KARATSUBA-PRODUCT(x,y,n)

1: **if** $x == 0 || y == 0$ **then**
2: **return** 0
3: **end if**
4: **if** $n == 1$ **then**
5: **return** $x \times y$
6: **end if**
7: $a = x/(10^{\frac{n}{2}})$
8: $b = x\%(10^{\frac{n}{2}})$
9: $c = y/(10^{\frac{n}{2}})$
10: $d = y\%(10^{\frac{n}{2}})$
11: $ac = KARATSUBA - PRODUCT(a, c, n/2)$
12: $bd = KARATSUBA - PRODUCT(b, d, n/2)$
13: $e = KARATSUBA - PRODUCT(a - b, d - c, n/2)$
14: $xy = ac * 10^n + e * 10^{\frac{n}{2}} + bd$
15: **return** xy

Karatsuba 乘法的时间复杂度

Karatsuba 乘法

$AB = ac \cdot 10^n + ((a-b)(d-c) + ac + bd) \cdot 10^{\frac{n}{2}} + bd$。

- 在计算两个 n 位整数相乘时，用到了 3 个 $\frac{n}{2}$ 位整数相乘,6 个 n 位整数加法以及 2 个移位操作。加法和移位操作的时间复杂度为 $O(n)$。

Karatsuba 乘法的时间复杂度

Karatsuba 乘法

$AB = ac \cdot 10^n + ((a-b)(d-c) + ac + bd) \cdot 10^{\frac{n}{2}} + bd$。

- 在计算两个 n 位整数相乘时，用到了 3 个 $\frac{n}{2}$ 位整数相乘,6 个 n 位整数加法以及 2 个移位操作。加法和移位操作的时间复杂度为 $O(n)$。
- 设时间复杂度为 $T(n)$，则有如下递推式：

$$T(n) = \begin{cases} O(1), n = 1 \\ 3\,T(\frac{n}{2}) + O(n), n > 1 \end{cases}$$

Karatsuba 乘法的时间复杂度

递推式：

$$T(n) = \begin{cases} O(1), n = 1 \\ 3\,T(\frac{n}{2}) + O(n), n > 1 \end{cases}$$

- 法一：直接算
- 法二：主定理

法一：直接算

令 $n = 2^k$。

$$
\begin{aligned}
T(n) &= 3\,T(\frac{n}{2}) + O(n) \\
&= 3^2\,T(\frac{n}{2^2}) + (3^1 + 3^0)\,O(n) \\
&= 3^3\,T(\frac{n}{2^3}) + (3^2 + 3^1 + 3^0)\,O(n) \\
&\cdots\cdots \\
&= 3^k\,T(1) + \frac{3^{k+1} - 1}{2}\,O(1) \\
&= (\frac{5}{2}(2^{\lg 3})^k - \frac{1}{2})\,O(1) \\
&= (\frac{5}{2}(2^{k \lg 3}) - \frac{1}{2})\,O(1) \\
&= (\frac{5}{2} n^{\lg 3} - \frac{1}{2})\,O(1) \\
&= O(n^{\lg 3})
\end{aligned}
$$

法二：主定理

假设有递推关系式 $T(n)=aT\left(\frac{n}{b}\right)+f(n)$，其中 n 为问题规模，a 为递推的子问题数量，$\frac{n}{b}$ 为每个子问题的规模（假设每个子问题的规模基本一样），$f(n)$ 为递推以外进行的计算工作。

a≥1，b>1为常数，f(n) 为函数，T(n) 为非负整数。则有以下结果（分类讨论）：

（1）若 $f(n)=O\left(n^{log_b a-\varepsilon}\right),\varepsilon>0$, 那么 $T(n)=\Theta\left(n^{\log_b a}\right)$

（2）若 $f(n)=\Theta\left(n^{\log_b a}\right)$, 那么 $T(n)=\Theta\left(n^{\log_b a}\log n\right)$

（3）若 $f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right),\varepsilon>0$, 且对于某个常数 $c<1$ 和所有充分大的 n 有 $af\left(\frac{n}{b}\right)\le cf(n)$, 那么 $T(n)=\Theta(f(n))$.

- 由 $O(n)=O(n^{\log_2 3-\epsilon})$，根据主定理第一条得：

$$T(n)=O(n^{\lg 3})$$

Contents

Toom-Cook 乘法

- 是 Karatsuba 乘法一般化以后的算法

Toom-Cook 乘法

- 是 Karatsuba 乘法一般化以后的算法
- Karatsuba 乘法是把 n 位整数分成 2 段计算，而 Toom-Cook 乘法是把 n 位整数分成 k 段计算，然后运用数学技巧尽量减少乘法次数。

Toom-Cook 乘法

- 是 Karatsuba 乘法一般化以后的算法
- Karatsuba 乘法是把 n 位整数分成 2 段计算，而 Toom-Cook 乘法是把 n 位整数分成 k 段计算，然后运用数学技巧尽量减少乘法次数。
- Toom-3 是 Toom-Cook 乘法的一个特例，即把 n 位整数分成 3 段计算。按照一般算式需要计算 9 个乘法，而运用数学技巧转化后可以只计算 5 个乘法，将时间复杂度降到了 $O(n^{\frac{\lg 5}{\lg 3}}) \approx O(n^{1.46})$。
- $(O(n^{\lg 3}) \approx O(n^{1.58}))$

Toom-Cook 乘法

- 是 Karatsuba 乘法一般化以后的算法
- Karatsuba 乘法是把 n 位整数分成 2 段计算，而 Toom-Cook 乘法是把 n 位整数分成 k 段计算，然后运用数学技巧尽量减少乘法次数。
- Toom-3 是 Toom-Cook 乘法的一个特例，即把 n 位整数分成 3 段计算。按照一般算式需要计算 9 个乘法，而运用数学技巧转化后可以只计算 5 个乘法，将时间复杂度降到了 $O(n^{\frac{\lg 5}{\lg 3}}) \approx O(n^{1.46})$。
- $(O(n^{\lg 3}) \approx O(n^{1.58}))$
- 但是具体的数学转化很复杂，感兴趣的同学可以 wiki 一下 Toom–Cook multiplication。

FFT & FFNT

- FFT：快速傅里叶变换。FFNT：快速数论变换。

FFT & FFNT

- FFT：快速傅里叶变换。FFNT：快速数论变换。
- 算法导论书上有句话就是说的这个算法：

另一个长整数的高或余数需要耗时 $\Theta(\beta^2)$。（见练习 31.1-12。）如今，人们已经有了更快的计算方法。例如，一个简单的分治算法可以在两个 β 位整数相乘的问题上达到 $\Theta(\beta^{\lg 3})$ 的运行时间。而已知最快的算法则只需要 $\Theta(\beta \lg \beta \lg \lg \beta)$ 的运行时间。然而在实际问题中，$\Theta(\beta^2)$ 的算法往往效

FFT & FFNT

- FFT：快速傅里叶变换。FFNT：快速数论变换。
- 算法导论书上有句话就是说的这个算法：

另一个长整数的高或余数需要耗时 $\Theta(\beta^2)$。（见练习 31.1-12。）如今，人们已经有了更快的计算方法。例如，一个简单的分治算法可以在两个 β 位整数相乘的问题上达到 $\Theta(\beta^{\lg 3})$ 的运行时间。而已知最快的算法则只需要 $\Theta(\beta \lg \beta \lg\lg \beta)$ 的运行时间。然而在实际问题中，$\Theta(\beta^2)$ 的算法往往效

- 一看名字就很难，再看时间复杂度就更难

FFT & FFNT

- FFT：快速傅里叶变换。FFNT：快速数论变换。
- 算法导论书上有句话就是说的这个算法：

另一个长整数的高或余数需要耗时 $\Theta(\beta^2)$。（见练习 31.1-12。）如今，人们已经有了更快的计算方法。例如，一个简单的分治算法可以在两个 β 位整数相乘的问题上达到 $\Theta(\beta^{\lg 3})$ 的运行时间。而已知最快的算法则只需要 $\Theta(\beta \lg \beta \lg\lg \beta)$ 的运行时间。然而在实际问题中，$\Theta(\beta^2)$ 的算法往往效

- 一看名字就很难，再看时间复杂度就更难
- 并没有搞懂的我肯定讲不了，感兴趣的同学可以 wiki 一下 Schönhage–Strassen algorithm。

最后的话

- 算法导论上有一句话：

而已知最快的算法则只需要 $O(p \lg p \lg \lg p)$ 的运行时间。然而在实际问题中，$\Theta(\beta^2)$ 的算法往往效果最好。我们也将以该算法作为算法分析的基准。

最后的话

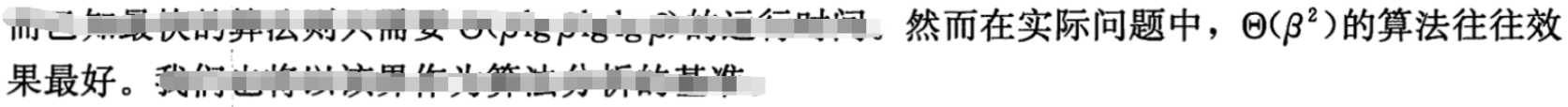
- 算法导论上有一句话：

  [redacted]。然而在实际问题中，$\Theta(\beta^2)$的算法往往效果最好。[redacted]

- 一开始的我太天真

最后的话

- 算法导论上有一句话：



- 一开始的我太天真
- 感兴趣的同学可以自己写一下呀。（挖坑


THANK YOU