Algorithm and Matrix

本短文讲述两件事。第一件是简介算法Algorithm一词的由来，面向广大的受众。第二部分需要一定的线性代数知识，将运用矩阵特征值等价地计算Fibonacci数列的通项。

# Algorithm, Algorism, and Algebra

**Algorithm**一词是纪念九世纪的波斯穆斯林数学家花拉子米Al-Khwarizmi（全名Abu Abdullah Muhammad ibn Musa Al-Khwarizmi阿布阿卜杜拉穆罕穆德伊本花拉子米）。

花拉子米在一本书中总结了十进制数字系统中执行算术运算（甚至计算π的小数部分）的基本规则，叫做**Algorism**。这些计算过程是如此的精确、无歧义、机械和高效，以至于到了十八世纪的欧洲，数学家们把花拉子米名字Al-Khwarizmi的拉丁文翻译Algorithm当作具有确定性过程的代名词。每当提到algorithm，他就是Al-Khwarizmi。

花拉子米的书原名叫*“Kitab al-jabr wa al-muqabalah”*（大意为*“Rules of Reintegration and Reduction”*），这本书为他赢得了代数学之父的美誉。其中代数**Algebra**就源于该书名中的”al-jabr”。

*题外话1*：作为计算机数字基础的二进制代数系统，是由英国人George Boole发明的。C++和Java等程序设计语言中的布尔类型bool（或者boolean）就是纪念这个人。

*题外话2*：算法概念的形式化（计算模型）直到二十世纪三四十年代才由图灵（Alan Turing）的图灵机和丘奇（Alonzo Church）的lambda演算完成，这也成为了计算机科学的基础。现在广为人知的”Church-Turing Thesis”是由丘奇的学生Stephen Kleene在1943年首先定义的。

# Algorithm, Fibonacci, and Matrix

事实上，如果没有Leonardo Fibonacci先生，花拉子米的工作在欧洲也许就没有那么稳固的立足之地了。Fibonacci是十五世纪的意大利数学家，Leonardo更为人熟知的是以他名字命名的Fibonacci数列（以下简称F数列）：

F0 = 0, F1 = 1, 1, 2, 3, 5, 8, 13, 21, … , F30 = 1346269, …

递归表达式如下：

F(n) = F(n-1) + F(n-2), n>1

F数列的增长有多快呢。让我们来算算，F31 = 1346269，就超过了一百万！实际上，F数列增长的几乎与2的幂次一样快：



如何计算Fn呢。最简单的方法就是按照其数学定义即可：

function fib1(n):

if(n==0) return 0;

if(n==1) return 1;

return fib1(n-1) + fib2(n-2);

这是一个递归程序，时间复杂度是指数级别：

T(n) = T(n-1) + T(n-2) + 3 > F(n)

*（其中T(n)表示当输入规模为n需要的时间，时间度量以一步基本操作计算，如3代表两步if语句判断操作加上一步加法操作）*

有没有更快的计算方法呢。注意到，上述递归算法有很多重复计算的步骤，比如计算F(5)是计算了一遍F(3)，而F(4)时又计算了一遍F(3)。一个更好的方法是存储已经计算好的F(i)，当需要用到它时直接读取即可。

function fib2(n):

if(n==0) return 0;

if(n==1) return 1;

*Allocate an array f[0…n]*

f[0] = 0; f[1] = 1;

for( i = 2…n ){

f[i] = f[i-1] + f[i-2];

}

return f[n];

上述算法用线性的空间将指数时间降到了多项式时间。如果将加法操作看作是一个基本操作（意味着单位时间内可以完成），则时间复杂度是线性的。但是考虑到F数列增长极快，而大整数的加法不是一个基本操作，而是线性的，因此上述算法更接近于二次复杂度。

还有更好的算法吗。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

你可能依稀记得F数列有一个通项公式：



直接带入公式计算不是更快吗！

等等，别急。F数列是一个自然数的序列，而上述公式中竟然出现了无理数，太神奇了。对于有限精度的计算机来说，这不是一个好消息。这个时候，矩阵计算粉墨登场了！

首先，将F数列以矩阵记号的形式重新表述，F1 = F1，F2 = F0 + F1 ：



一般地：



因此，求F(n)的过程，等价于迭代计算整系数矩阵的n次幂。

现在推导（E2）与（E1）等价。

计整系数矩阵为A，即：



A的特征多项式：



得到A的特征根组成的对角矩阵D:



对角化A的可逆矩阵P由A的特征向量构成：



由线性代数中关于矩阵对角化的知识可知：



将上述结果带入（E2）立即可得（E1）。

*注记1*：矩阵方法依赖于快速矩阵乘法，复杂度低于二次。

*注记2*：在参考资源3中还指出了Fibonacci数列与黄金比例的关系。

# 参考资源

1. History of Algorithms and Algorithmics:

<http://www.scriptol.com/programming/algorithm-history.php>

1. Where the Word “Algebra” Came From:

<http://www.todayifoundout.com/index.php/2010/12/the-origins-of-the-word-algebra/>

1. Fibonacci Numbers：

<http://www.mathworks.com/moler/exm/chapters/fibonacci.pdf>