# 欧拉项目趣题分析
# The Key to Project Euler’s Problems

A
SIMPLE OVERVIEW
TO
PROJECT EULER’S PROBLEM

Masters

21st Century

[0]All are provided or inspired by ProjectEuler.net。

所有问题编号皆按难度升序排列。

文中插入的代码可能与原 PDF 的稍有不同，敬请原谅。

**PDF 信息**

制作者　Jak Wings
联系方式　j_kylin _AT_ 126.com
制作工具　XeTeX
更新日期　July 11, 2011

# 关于欧拉项目

该文档来译自 ProjectEuler.net

**什么是欧拉项目?**

欧拉项目提供了一系列富有挑战性的数学/计算机编程问题，要解答这些问题不是单纯懂数学就可以的了。虽然通过数学能令助你想出优雅而高效的解决方法，不过多数问题还是要依靠计算机以及对编程技巧的应用。

之所以要创建欧拉项目，并持续发展它，是为了给那些好学的人提供一个趣味的平台，去深入探究陌生的领域，并学习新奇的理念。

**这些问题为谁而设?**

这些问题面向的对象包括：一，那些在基础课程上求知欲得不到满足的学生；二，虽不从事数学相关的工作但对数学感兴趣的人；三，那些想保持它们的数学解题能力的专家们。

**谁都有能力解决这些问题吗?**

这些问题的难度不一，但提供了一条经验归纳的学习链。即是说，通过解决一道问题，一个新的想法就会展示在你面前，并帮助你解决以前不懂的题目。因此，只要坚持解答这些问题的话，慢慢地你就会懂得如何解答每一个问题。

**我该从哪道题开始呢?**

这取决于你的智力。你可以在问题列表中看到每个问题分别被多少人解决过，显然，越多人解决过的问题就越简单，你可以从简单题目的开始做。

**我已经写好了解答题目的程序，不过要不要花几天时间才算出答案呢?**

绝对不应该这样做！每一个问题都是精心设计的，并遵循“一分钟原则”。也就是说，即使你要花几小时去设计一个成功的算法去解决一个难题，但运用了这个算法的程序运行在一般配置的计算机上应该在一分钟之内得到结果。

**假如我的程序花了超过一分钟的时间得到问题的答案，是否介意呢?**

当然不介意，不过你应该再想想如何才能改进你的算法并缩短解题时间。另外请注意，一旦你解决了某个问题，你就能去到这个问题的讨论贴，在那里你能从那些解决了这个问题的人身上了解到一些不同的解题方法。

**我通过搜索引擎解决了问题，这有关系么?**

我们提倡你通过搜索引擎去研究这些问题，因为在这些问题的表面之下可能潜藏大量的数学精髓。然而，我们要分清“探讨方法”和“利用在其它网站上搜得的答案”，如果你直接把搜索到的答案复制提交，你还能得到提高吗?

**我检查我的程序很多遍了，但网站一直说我提交的答案是错误的！你们是不是搞错了什么?**

对于新发表的问题来说，的确很有可能是因为通过网络发表时出现了一些小问题，或者是问题的描述有点模糊且没被解释清楚。然而若很多人都解决了问题，且你运行了十次程序也得不到正确答案，你总不能说这是网站的问题吧?

**能提供一些解题的提示么?**

请仔细地审题并记下提供的例子，然后在草稿纸上推算一下，看能不能发现问题的解决思路。如果解题思路对你来说有点陌生，不妨通过网络或书籍获取相关信息，问题本身应该能告诉你该搜寻什么资料了。

试着写个程序去解决问题的简单情况，并对照提供的例子来看算得的结果是否与之相符，这样你就知道你否接近正确的解答方法了。估计你的程序要计算出正确答案需要花多少时间，如果会超过一分钟的话，请重新构思解题方式。

**这一切是如何开始的?**

欧拉项目是由 Colin Hughes，即欧拉（euler），在 2001 年十月发起的一个在 mathschallenge.net 上的小栏目。谁也没料到这些问题会变得如此受欢迎呢？它的会员数越来越多，自此以后欧拉项目在 2006 年在新的独立域名上安家了。

**欧拉项目是由谁来维护的?**

新问题的灵感皆来自我们的会员，并且这些问题都是一群优秀的数学家和程序员的辛勤劳动的结果。故简单地说，欧拉项目是由全体会员共同维护的。

**我能资助这个网站吗?**

无任欢迎！欧拉项目完全由它的会员资助，故如果你喜欢这些问题并想帮助我们减轻维护经费的话，请自愿捐赠吧。（PayPal 捐赠按钮在其主页上）

**“欧拉项目，将鼓励和挑战所有热爱数学的人，令其在乐趣中提高能力。”**

(*"Project Euler exists to encourage, challenge, and develop the skills and enjoyment of anyone with an interest in the fascinating world of mathematics."*)

# Contents

# Chapter 1

# 求 1000 以下所有是 3 或 5 的倍数的自然数的和

一个简单的方法就是遍历所有从 1 到 999 的自然数，并逐个测试它们是否能被 3 或 5 整除。伪代码将如下：

```
001  target = 999
002  sum = 0
003  for i = 1 to target do
004      if (i mod 3 == 0) or (i mod 5 == 0) then sum = sum + i
005  output sum
（在某些编程语言中，mod 会被%替代。）
```

嗯，你也认为这实在太简单了吧?

不过请等一下：如果题目要求我们算 10,0000,0000 以内的数字的话，这真得等上好一会才能得到结果了。也许你还要检查一下，看数值有没有溢出（超出最大数字储存）。

要得到一个为之更有效率的解决方式，你应该计算一下 1000 以下能被 3 整除的自然数的个数，还有能被 5 整除的数字的个数。不过这样你就把能被 15 整除的数的个数统计了两遍了，故你还要减去能被 15 整除的数的个数。

我们可定义一个函数：

```
001  Function SumDivisibleBy (n)
002      函数内容
003  EndFunction
```

于是答案将是：

$$SumDivisibleBy(3) + SumDivisibleBy(5) - SumDivisibleBy(15)$$

[0]Author: hk

现在来具体分析这个函数，且假设 $n = 3$。
求和：

$$3 + 6 + 9 + 12 + \ldots + 999 = 3 \times (1 + 2 + 3 + 4 + ... + 333)$$

假设 $n = 5$：

$$5 + 10 + 15 + \ldots + 995 = 5 \times (1 + 2 + ... + 199)$$

（注意，333 和 199 分别是 999/3 和 999/5 向下取整的结果。在许多编程语言里，向下取整除法的运算符是 div 或 /。）

假如你知道 $1 + 2 + 3 + \ldots + n = \frac{n \times (n+1)}{2}$，我们的程序将会是：

```
001  target = 999
002
003  Function SumDivisibleBy (n)
004      p = target div n
005      return n*(p*(p+1)) div 2
006  EndFunction
007
008  Output SumDivisibleBy(3)+SumDivisibleBy(5)-SumDivisibleBy(15)
```

# Chapter 2

# 求不超过 400,0000 的 Fibonacci 数列中所有偶数的和

我们可以根据题目直接写出类似下面的程序：

```
001  limit = 4000000
002  sum = 0
003  a = 1
004  b = 2
005  while b <= limit
006      if b mod 2 == 0 then sum = sum + b
007      next = a + b
008      a = b
009      b = next
010  output sum
```

现在，让我们看看能不能免去判断每一项是否为偶数。

如下为该 Fibonacci 数列的前几项，偶数以红色标示：

| 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|
| c | a | b | c | a | b | c | a | b | c | a | … |

我们可以很容易证明每两个是偶数的 Fibonacci 数都间隔 2 个项。

于是我们就可以稍微改造一下之前的程序，使得每隔三项就累加一次：

```
001  limit = 4000000
002  sum = 0
003  a = 1
004  b = 2
005  c = a + b
006  while b <= limit
007      sum = sum + b
008      a = b + c
009      b = c + a
010      c = a + b
011  output sum
```

[0]Author: hk

其实在这个问题下还潜藏着一个美妙的构造。

如果我们只写出该 Fibonacci 数列的中的偶数：

2 8 34 144 ⋯

这个新的数列似乎符合该递归关系式：$E(n) = 4\,E(n-1) + E(n-2)$。

假如我们能证明对原来的 Fibonacci 数列 $F(n) = 4\,F(n-3) + F(n-6)$ 成立，我们也就证明了以上的的递归算式。

证明方式如下，不过你可能希望在继续阅读前尝试独自证明：

$$\begin{aligned} F(n) &= F(n-1) + F(n-2) \\ &= F(n-2) + F(n-3) + F(n-2) = 2F(n-2) + F(n-3) \\ &= 2(F(n-3) + F(n-4)) + F(n-3) \\ &= 3F(n-3) + F(n-4) + F(n-5) + F(n-6) \\ &= 4F(n-3) + F(n-6) \end{aligned}$$

在关于这个问题的论坛主题里，还有几种能避免判断 Fibonacci 数的每一项是否为偶数的方法。如果你在该文档找不到你的方法，你可以去那里找找看。

# Chapter 3

# 求和的平方与平方和之间的差

首先，谁都可能会想到用暴力算法，因为 100 实在不是一个很高的限制，于是程序很快就出来了：

```
001  limit = 100
002  sum_sq = 0
003  sum = 0
004  for i = 1 to limit do
005      sum = sum + i
006      sum_sq = sum_sq + i*i
007  print sum*sum - sum_sq
```

然而，这样的算法在要计算的数字很多的时候就显得有点马虎了。

再仔细想想这个程序，我们知道变量 $sum$ 保存了从整数 1 到 $limit$ 的和。众所周知的是，这个和可直接通过这条公式算出：$sum(n) = \frac{n(n+1)}{2}$。也许你也会想在平方和的算式中找到这类公式，那么就来找找看吧。

现在，我们要寻找这么一个函数 $f(n)$，给出正整数 n 就能求得整数 1 到 n 的平方和。我们不妨假设它的一般式为 $f(n) = an^3 + bn^2 + cn + d$，我们要确定 a、b、c、d 四个常数的值。这很容易求得，因为已知 $f(0) = 0,\ f(1) = 1,\ f(2) = 5,\ f(3) = 14$，这样就能得到一个四元一次方程组，即有：

$$\begin{aligned} d &= 0 \\ a+b+c+d &= 1 \\ 8a+4b+2c+d &= 5 \\ 27a+9b+3c+d &= 14 \end{aligned}$$

解如上方程组得 $a = \frac{1}{3}, b = \frac{1}{2}, c = \frac{1}{6}, d = 0$，于是有 $f(n) = \frac{1}{6}(2n^3 + 3n^2 + n) = \frac{n(n+1)(2n+1)}{6}$。

接下来我们要做的就是证明函数 $f$ 的确是我们想要找的公式。用数学归纳法，假设 $f$ 对于前 $n$ 个正整数成立，接着我们要证明它对前 $n+1$ 个正整数也成立。于是，已知当 $n = 0,\ 1,\ 2,\ 3$ 时，$f(n)$ 都成立，然后我们要证明

[0]Author: Lord Farin

$f(n+1) = f(n) + (n+1)^2$，通过展形该等式可得到：

$$\begin{aligned} f(n+1) &= f(n) + (n+1)^2 \\ \frac{n^3}{3} + \frac{3n^2}{2} + \frac{13n}{6} + 1 &= \frac{n}{6} + \frac{n^2}{2} + \frac{n^3}{3} + n^2 + 2n + 1 \end{aligned}$$

等式两边相等，故证明 $f$ 就是我们要找的公式。这样，我们现在就能写一个非常简单的程序去计算和的平方与平方和的差了：

```
001  limit = 100
002  sum = limit * (limit + 1) / 2
003  sum_sq = limit * (limit + 1) * (2*limit + 1) / 6
004  print sum*sum - sum_sq
```

这个算法只有一个限制，那就是你的编程语言（和计算机内存）所能表示的整型变量的大小。

# Chapter 4

# 求能被从 1 到 20 的自然数整除的最小正整数

其实这就相当于要算出从 2 到 20 的所有自然数的最小公倍数。给出的要除的数 $k$ 的上限 20，实在太小了，我们可以轻松地把答案笔算出来。不过我们得设计一个高效的通用算法来解决上限比 20 更大的情况。

假设 $N$ 是能被 2 到 k 整除的最小正整数，根据 $N$ 的性质，可知 $N$ 的质因式分解中不会包含多余的素数，除非它是必需的。

思考一下 $k$ 的前三种情况:

$$k = 2,\ N = 2$$
$$k = 3,\ N = 2 \times 3 = 6$$
$$k = 4,\ N = 2 \times 3 \times 2 = 12$$

可以看出，当 $k = 4$ 时，我们不必算 $2 \times 3 \times 4$，因为因式分解 $4 = 2 \times 2$ 中需要的一个 2 早就存在了。若我们再思考下接下来的两种情况:

$$k = 5,\ N = 2 \times 3 \times 2 \times 5 = 60$$
$$k = 6,\ N = 2 \times 3 \times 2 \times 5 = 60$$

可见 $k = 5$ 和 $k = 6$ 两种情况下 $N$ 都为 20，因为若 $N$ 有质因数 2 和 3，那么它就能被 6 整除。

根据这个原理，当 $k = 20$ 时:

$$N = 2 \times 3 \times 2 \times 5 \times 7 \times 2 \times 3 \times 11 \times 13 \times 2 \times 17 \times 19 = 232792560$$

那该如何通过编程解决这个问题呢?

不妨假设 $p[i]$ 是第 p 个素数，则 $p[1] = 2,\ p[2] = 3,\ p[3] = 5,\ \ldots$。
再设 $N = p[1]^{a[1]} \times p[2]^{a[2]} \times p[3]^{a}[3] \times \ldots$。
一开始，令 $a[i] = 0,\ i = 1,\ 2,\ 3,\ \ldots$。
$For j = 2 to k$，求 $j$ 的质因式分解 $p[1]^{b}[1] \times p[2]^{b}[2] \times p[3]^{b}[3] \times \ldots$ 并使

[0] Author: euler

$a[i] = max(a[i], b[i])$。

现在来看看这个算法处理 $k$ 直到 10 的详细情况：

| j | b[1] | b[2] | b[3] | b[4] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | | | | 1 | | | |
| 3 | | 1 | | | 1 | 1 | | |
| 4 | 2 | | | | 2 | 1 | | |
| 5 | | | 1 | | 2 | 1 | 1 | |
| 6 | 1 | 1 | | | 2 | 1 | 1 | |
| 7 | | | | 1 | 2 | 1 | 1 | 1 |
| 8 | 3 | | | | 3 | 1 | 1 | 1 |
| 9 | | 2 | | | 3 | 2 | 1 | 1 |
| 10 | 1 | | 1 | | 3 | 2 | 1 | 1 |

于是现在 $N = 2^3 \times 3^2 \times 5^1 \times 7^1 = 2520$。

然而，这个算法需要设计一个函数去求每个数的质因式分解，这太麻烦了，因此我们又得想想有更好的方法。

考虑下当 $k = 20$ 时 $N$ 的值，已知 $N$ 必定能被所有小于或等于 $k$ 的质数整除，但是该如何确定 $N$ 的质因式分解中每个质数的幂 $a[i]$ 呢？不难发现这是由小于 $k$ 的 $p[i]$ 的完全一次或多次方数决定的。例如，知道 $2^4 = 16$ 且 $2^5 = 32$，我们就知道 $a[1] = 4$ 且 2 到 20 中总共包涵 4 个或以上的因数 2。同样地，只要知道 $3^2 = 9$ 且 $3^3 = 27$，就知道 $a[2] = 2$。若 $p[i] \geq 5$，则 $a[i] = 1$。

因此 $N = 2^4 \times 3^2 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 = 232792560$。

给出了质因数 $p[i]$，可以这样求出它的幂 $a[i]$：

假设 $p[i]^{a[i]} = k$，对该等式两边同时取对数得：$a[i]\log(p[i]) = \log(k)$。
因此，$a[i] = \frac{\log(k)}{\log(p[i])}$。
已知 $a[i]$ 是整数，故 $a[i] = \lfloor \frac{\log(k)}{\log(p[i])} \rfloor$。

例如，若 $k = 20$，那么它的第一个质因数 $p[1] = 2$ 的幂为：

$$a[1] = \lfloor \frac{\log(20)}{\log(2)} \rfloor = \lfloor 4.32 \rfloor = 4$$

为了优化这个算法，我们知道当 $p[i]^2 > k$ 时 $a[i] = 1$，即是说只有当 $p[i] \leq \sqrt{k}$ 时才需要算出 $a[i]$。

将以上的所有思路整合成一个完整的算法，我们还必须先找到 $N$ 的所有质因数 $p[1]$, $p[2]$, $p[3]$, ...。

```
001  k = 20
002  N = 1
```

```
003  i = 1
004  check = true
005  limit = sqrt(k)
006  while p[i] <= k
007      a[i] = 1
008      if check then
009          if p[i] <= limit then
010              a[i] = floor(log(k) / log(p[i]))
011          else
012              check = false
013          end if
014      end if
015      N = N * p[i]^a[i]
016      i = i + 1
017  end while
018  output N
```

必须指出，这个算法实际上受到的限制主要是你的编程语言能处理的数字 $N$ 的大小范围，当 $k$ 增大的时候，$N$ 的大小会以几何级的数量递增。

# Chapter 5

# 求一个合数的最大质因数

假设有一个合数 $n$，且 $k = 2, 3, 4, 5, \ldots$。对于每一个 $k$，若它是 $n$ 的因数，那么就用 $k$ 去除 $n$，直到无法再整除时才用下一个 $k$ 去除。可以看出，若 $k$ 是因数，那么当所有更小的因数都用过之后，它最终肯定是一个质因数，并且最终整除得到的商必是 1。

这样，我们就得到了一个很简陋的算法：

```
// 如果你无法对变量赋这么大的值的话，可以参见本章节最后一页的提示。
001  n = 一个“狠”大的数
002  factor = 2
003  lastFactor = 1
004  while n > 1
005      if n mod factor = 0 then
006          lastFactor = factor
007          n = n div factor
008          while n mod factor = 0
009              n = n div factor
010      factor = factor + 1
011  output lastFactor
```

虽然这个算法能完成任务了，不过我们还是能改进一下的。

首先，2 是唯一的偶质数，因此只要分开处理质因数是 2 的情况，就可以在每一次循环中使因数 $factor$ 递增 2：

```
001  n = 一个“狠”大的数
002  if n mod 2 = 0
003    then
004      lastFactor = 2
005      n = n div 2
006      while n mod 2 = 0
007          n = n div 2
```

[0]Author: hk

```
008    else
009      lastFactor = 1
010  factor = 3
011  while n > 1
012      if n mod factor = 0
013        then
014          lastFactor = factor
015          n = n div factor
016          while n mod factor = 0
017              n = n div factor
018      factor = factor + 2
019  output lastFactor
```

不过题目中要让你求的最大的质因数还不是很大。

假如现在要进行因式分解的数字是 $2 \times 1009\times$ (一些大质数) 的话，那么要得到最大的质因数就要循环好多步才行了。不过利用以下重大发现，这个算法又可以大大改进了:

> 一个正整数的最大质因数不大于 $\sqrt{n}$。

于是每次从 $n$ 的因数中约去一个质因数后，用商的平方根来作为下一个因数 $factor$ 的上限，若 $factor$ 大于这个上限，那么这个商就是质因数。

改进后的算法如下:

```
001  n = 一个“狠”大的数
002  if n mod 2 = 0
003    then
004      lastFactor = 2
005      n = n div 2
006      while n mod 2 = 0
007          n = n div 2
008    else
009      lastFactor = 1
010  factor = 3
011  maxFactor = sqrt(n)
012  while n > 1 and factor <= maxFactor
013      if n mod factor = 0
014        then
015          lastFactor = factor
016          n = n div factor
017          while n mod factor = 0
018              n = n div factor
019          maxFactor = sqrt(n)
020      factor = factor + 2
021  if n = 1
022    then
023      output lastFactor
024    else
025      output n
```

下面，就是关于如何使变量能表示出大数的问题了。

题目中给出的数字 6008,5147,5143 已经大大超过 $2^{31}-1$ 即 21,4748,3647 了。

像 C、Pascal、Visual Basic 这些语言中的数据类型 int 或 integer 一般都是 32 位有符号数，且无法处理大于 $2^{31}-1$ 的整数。

因此你必须用别的数据类型来处理这么大的数字，一般有 long、long long、int64、_int64 或 __int64 等。如果你把变量定义为这些类型，或许你还得在数字上加个后缀，如 600851475143L。

由于这是由编程语言和编译环境决定的，故你最好还是查看相关的数据类型说明文档。以后你还会遇到要求使用大数的问题的。

另外还得注意的是，当你求两个数的和或积的时候，结果也可能超出 32 位有符号整型变量能处理的数字的范围。这时常说的“类型转换”可能会有所帮助。

可是，这份问题分析并不是什么编程教程，你得自己先去了解好你学习的编程语言。在许多情况下，请求对一个问题的帮助是不必要的，假如你已经事先了解到足够的信息。

# Chapter 6

# 求由两个 3 位数的积构成的最大回文数

假设该回文数是 $P = ab$，$a$ 和 $b$ 都是三位数，那么 $a$ 和 $b$ 都是 100 到 999 内的数字。于是最开始我们的程序可能就是：

```
001  function reverse (n)
002      reversed = 0
003      while n > 0
004          reversed = 10 * reversed + n mod 10
005          n = n / 10
006      return reversed
007
008  function isPalindrome (n)
009      return n = reverse(n)
010
011  largestPalindrome = 0
012  a = 100
013  while a <= 999
014      b = 100
015      while b <= 999
016          if isPalindrome(a*b) and a*b > largestPalindrome
017              largestPalindrome = a * b
018          b = b + 1
019      a = a + 1
020  output largestPalindrome
```

对于题目给出的情况来说，这样已经够快了，不过还可应该改进一下。

首先，这种方法进行了许多重复的判断。例如，69696 这个数出现的情况有两种，一种是当 $a = 132,\ b = 528$ 时，另一种是当 $a = 528,\ b = 132$ 时。要避免这种重复的判断，我们可以加个约束 $a \leqslant b$，这样就大概减少了一半的计算量。于是程序会变成这样：

```
// 上半部分定义的函数不变
```

[0]Author: Lster

```
010
011  largestPalindrome = 0
012  a = 100
013  while a <= 999
014      b = a // 替换原来的 b = 100
015      while b <= 999
016          if isPalindrome(a*b) and a*b > largestPalindrome
017              largestPalindrome = a * b
018          b = b + 1
019      a = a + 1
020  output largestPalindrome
```

接下来，我们还可以考虑改成从 999 算到 100，这样该程序就更可能迅速地找到最大的回文数了，这样我们就可以早一点退出循环，减少很多不必要的计算了。代码如下:

```
// 上半部分不变
010
011  largestPalindrome = 0
012  a = 999
013  while a >= 100
014      b = 999
015      while b >= a
016          if a*b <= largestPalindrome
017              break //因为以后的 a*b 会更小
018          if isPalindrome(a*b)
019              largestPalindrome = a * b
020          b = b - 1
021      a = a - 1
022  output largestPalindrome
```

这样程序就更快了。不过再加多一点分析的话，速度还能大大提升呢。

我们要求的回文数 $P$ 必定是 6 位数，因为有 $111111 = 143 \times 777$。然后再假设 $P$ 的前三位数分别是 $x$、$y$、$z$，由于 $P$ 是回文数，故有:

$$\begin{aligned} P &= 100000x + 10000y + 1000z + 100z + 10y + x \\ &= 100001x + 10010y + 1100z \\ &= 11(9091x + 910y + 100z) \end{aligned}$$

既然 11 是一个质数，那么 $a$ 和 $b$ 中至少有一个有因数 11，故假如 $a$ 不能被 11 整除的话，$b$ 就肯定可以。利用这一点，我们就可以根据 $a$ 来决定如何检查 $b$ 的值了。

```
// 上半部分的函数不变
010
011  largestPalindrome = 0
012  a = 999
013  while a >= 100
014      if a mod 11 = 0
015        then
016          b = 999
017          step = 1
018        else
019          b = 990
020          step = 11
021      while b >= a
022          if a*b <= largestPalindrome
023              break
024          if isPalindrome(a*b)
025              largestPalindrome = a * b
026          b = b - step
027      a = a - 1
028  output largestPalindrome
```

# Chapter 7

# 求第 10001 个质数

我们也不知道该怎么解决这个问题，因此我们将会用 **试除法**。然而，如果预先知道我们要求的质数的明确上限，用 **Eratosthenes 筛选法**将更为高效。

以下是一些有用的结论：

1 不是质数。
2 是唯一的偶质数。
所有大于 3 的质数都可表示成 $6k \pm 1$ 的形式，$k \in N^+$。
任一正整数 $n$ 都最多只有一个大于 $\sqrt{n}$ 的质因数。

因此，检查 $n$ 是否为质数的基本方法就是：如果我们无法找到能整除 $n$ 的一个小于或等于 $\sqrt{n}$ 的整数 $f$，那么 $n$ 就是质数，即 $n$ 的唯一的质因数就是 $n$ 自身。

让我们基于上述结论来设计检测 $n$ 是否为质数的算法吧：

```
001  Function isPrime (n)
002      if n = 1 then return false
003          else
004          if n < 4 then return true // 2 和 3 都是质数
005              else
006              if n mod 2 = 0 then return false
007                  else
008                  if n < 9 then return true // 已排除 4、6、8
009                      else
010                      if n mod 3 = 0 then return false
011                          else
012                          r = floor(sqrt(n)) // 向下取整
013                          f = 5
014                          while f <= r
015                              if n mod f = 0 then return false
016                             if n mod (f+2) = 0 then return false
017                              f = f + 6
018                          endwhile
019                          return true (其它所有情况)
```

[0]Author: hk

```
020  End Function
```

上面定义的是检测函数，然后用它来求出我们要的第 10001 个质数：

```
022  limit = 10001
023  count = 1 // 已知 2 为质数，预先统计
024  candidate = 1
025  repeat
026      candidate = candidate + 2
027      if isPrime(candidate) then count = count + 1
028  until count = limit
029  output candidate
```

# Chapter 8

# 求在某一 1000 位数中连续 5 个位上的数的积的最大值

为了避免大家对题目的理解有错误，我们试举一例。若有一个 10 位数 4124012983，要求其中连续 3 位数的积的最大值，那么连续三位数的积有：

$$\begin{aligned} 4 \times 1 \times 2 &= 8 \\ 1 \times 2 \times 4 &= 8 \\ 2 \times 4 \times 0 &= 0 \\ 0 \times 1 \times 2 &= 0 \\ \vdots \times \vdots \times \vdots &= \vdots \\ 9 \times 8 \times 3 &= 216 \end{aligned}$$

显然，连续三位数的最大积是 $216 = 9 \times 8 \times 3$。

那么，大家或许会认为，Euler Project 到底为什么要出这么简单的题呢? 直接用暴力算法就能轻松解决了。

循环 996 遍，把所有 1000 位数分成 996 种情况，保存下最大的乘积就行了。假设 1000 位数的每一位已预先读入了数组 $digits$ 中：

```
001  digits = 1000 位数字
002  limit = 1000
003  queue = 5
004  largestProduct = 0
005  tempProduct = 1
006  for i = 1 to limit - queue + 1 do
007      for j = 0 to queue - 1 do
008          tempProduct = tempProduct * digits[i+j]
009      if tempProduct > largestProduct then
010          largestProduct = tempProduct
011  output largestProduct
```

嗯，似乎代码够短够简单的了，简直没有难度。不过大家有没发现，每次连

[0]Author: Jak Wings

乘的时候数字都有重复使用，且能重复使用达 5 次之多！假如能避免过多的重复，就能省下很多的计算时间了。

为此，我们可以每次循环都除以上一次连乘中第一位数，然后乘以这次连乘中新出现的最后一位数，这样就达到了优化了：

```
001  digits = 1000 位数字
002  limit = 1000
003  queue = 5
004  tempProduct = 0
005  for i = 1 to queue do
006      tempProduct = tempProduct * digits[i]
007  largestProduct = tempProduct
008  for i = queue + 1 to limit do
009      tempProduct = tempProduct / digits[i-5] * digit[i]
010      if tempProduct > largestProduct then
011          largestProduct = tempProduct
012  output largestProduct
```

省去了很多重复的运算，程序快多了。

再仔细观察下，会发现连续五位数中若有 0，则乘积必然是最小的，故遇到 0 的时候可跳过，然后开重新计算连乘积。

还有一个优化手段就是，可以一边读取下一位数字，一边用上面的方法，暂且称之为“滚轮法”吧，这样就不用预先读取 1000 位数字到 $digits$ 中，实现读完即解。这里就不提供代码了，请大家独自思索吧。

# Chapter 9

# 求三个数的和为 1000 的勾股数组

无疑，勾股数组 $(a, b, c)$ 中 $a + b + c = 1000$ 只是任意一种情况，因此我们不如讨论更一般性的问题：找出所有勾股数组，其中 $a + b + c = s$。

似乎每组勾股数的和都为偶数，不过你或许想自己去证明一下。

## 9.1 直接的算法

最直接的算法就是简单地遍历 $a$ 和 $b$，并检查是否 $a^2 + b^2 = (s - a - b)^2$。根据 $a < b < c$，我们得到 $a \leqslant \frac{s-3}{3}$ 且 $b < \frac{s-a}{2}$。于是程序代码如下：

```
001  s := 1000
002  for a := 3 to (s - 3) div 3
003      for b := (a+1) to (s-a-1) / div 2
004          c := s - a - b
005          if c*c = a*a + b*b then
006              output (a, b, c)
007          end if
008      end for
009  end for
```

只要 $s$ 不是很大，那么以上的算法就足够快了，不过它的递增适应性并不是很好。如果你把 $s$ 变为原来的 $k$ 倍，那么每一层循环的次数就是原来的 $k$ 倍，由于两层循环是嵌套的，故总的检查次数就是原来的 $k^2$ 倍了。因此，若 $s$ 加倍了，则这个程序或许要花大概为原来 4 倍的时间来检查，若 $s$ 为原来的 10 倍，则花的时间就为原来的 100 倍左右了。

假如算出更精确的循环范围，这个算法的速度就会更快，不过它的递增适应性还是没变，对于 $s = 1000000$ 的情况来说，这个算法有点勉强了。

## 9.2 将勾股数组以参数化的形式表示

根据定义，当 $\gcd(a, b, c) = 1$（$a, b, c$ 的最大公约数为 1）时，勾股数组 $(a, b, c)$ 为朴素形式。若勾股数组 $(a, b, c)$ 符合 $\gcd(a, b) = \gcd(b, c) = \gcd(c, a)$，则它是朴素的当且仅当 $\gcd(a, b) = 1$。早在古希腊的人就已经知道，所有朴素的勾股数组都能表示为：

[0]Author: daniel.is.fischer

$$a = m^2 - n^2, \quad b = 2 \cdot m \cdot n, \quad c = m^2 + n^2$$
$$m > n > 0$$

或许还要交换一下 $a$ 和 $b$ 才能使 $a < b$。利用上面这个公式保证能得到一个勾股数组，不过它是朴素的，当且仅当 $m, n$ 中恰有一个是偶数，且 $\gcd(m, n) = 1$。

于是每个勾股数组都有唯一的表示方式：

$$a = (m^2 - n^2) \cdot d, \quad b = 2 \cdot m \cdot n \cdot n, \quad c = (m^2 + n^2) \cdot d \quad (*)$$
$$m > n > 0, \quad \gcd(m, n) = 1$$

以上，$m, n$ 中恰有一个是偶数，$d$ 是 $a, b, c$ 的最大公约数。

利用这个参数形式，可得：

$$a + b + c = 2 \cdot m \cdot (m + n) \cdot d$$

因此，要找到符合 $a + b + c = s$ 的勾股数组 $(a, b, c)$，我们就得找到能整除 $\frac{s}{2}$ 的 $m (> 1)$，和能整除 $\frac{s}{2m}$ 的 $k (= m + n)$，且 $\gcd(m, k) = 1$。于是就有 $n = k - m$, $d = \frac{s}{2mk}$，将之代入到 $(*)$ 中求得 $(a, b, c)$。

这个算法的一个相当简单的实现如下：

```
001  s2 := s div 2
002  mlimit := ceiling(sqrt(s2)) - 1
003  for m := 2 to mlimit
004      if s2 mod m = 0 then
005          sm := s2 div m
006          while sm mod 2 = 0      // 通过除去因数 2
007              sm := sm div 2      // 缩小寻找 k 的循环
008          end while
009          if m mod 2 = 1 then k := m + 2 else k := m + 1
010          while k < 2*m and k <= sm
011              if sm mod k = 0 and gcd(k, m) = 1 then
012                  d := s2 div (k*m)
013                  n := k - m
014                  a := d * (m*m - n*n)
015                  b := 2 * d * m * n
016                  c := d * (m*m + n*n)
017                  output (a, b, c)
018              end if
019              k := k + 2
020          end while
021      end if
022  end for
```

$ceiling(x)$ 即将对 $x$ 向上取整，可表示为 $\lceil x \rceil$，结果为大于 $x$ 的最小整数。这个算法寻找指定的勾股数组更快，递增适应性很好，当 $s \leqslant 10^{10}$ 时，也只需一秒不到的时间就算出结果了。然而，利用更复杂的代码，它还能加快一点点，只要知道 $\frac{s}{2}$ 的质因式分解就行了。

## 9.3 关于勾股数组的参数化

### 9.3.1 勾股数组是朴素勾股数组的充要条件的证明

让我们重新声明一下：

勾股数组 $(a,b,c)=(m^2-n^2,2mn,m^2+n^2),\ m>n>0$ 是朴素的，当且仅当 $\gcd(m,n)=1$，且 $m,n$ 中恰有一个是偶数。

若 $\gcd(m,n)=d>1$，则显然 $d^2$ 是 $a,b,c$ 的公约数，且若 $m,n$ 都为偶数，则 $a,b,c$ 都为偶数，故 $\gcd(m,n)=1$ 这个条件是必要的。

另一方面，假设 $m,n$ 中恰有一个是偶数，且这个勾股数组不是朴素的，令 $p$ 是能整除 $a,b,c$ 的质数。已知 $a$ 和 $c$ 是奇数，那么 $p$ 也是奇数。任何一个能整除 $b$ 的奇质数必然至少能整除 $m,n$ 中的一个，若 $p$ 能整除 $m$，则 $p$ 也能整除 $m^2$。可是，因为 $p$ 能整除 $c=m^2+n^2$，故 $p$ 也能整除 $n^2$（和 $n$）。这样，$\gcd(m,n)>1$。故以上的充要条件得证。

## 9.3.2 该参数化形式的推导过程

假设 $T=(a,b,c)$ 是一个勾股数组，且设 $x=\frac{a}{c},\ y=\frac{b}{c}$。那么由实数 $x,y$ 组成的坐标 $(x,y)$ 就是单位圆上的一点，所有通过将 $a,b,c$ 乘上同一的实数得到的勾股数组 $(a',b',c')$，在单位圆上都有同一点 $(x',y')$。反过来，若 $x=\frac{p}{q},\ y=\frac{r}{q}$ 都是正实数，且 $x^2+y^2=1$，则 $(p,r,q)$ 是一个勾股数组（不一定有 $a<b$）。

现在，考虑一条经过点 $(0,-1)$ 的具有正斜率 $s$ 的直线，这条线的的参数形式为 $(t,s\cdot t-1),\ t\in R$，这条直线和单位圆的另一交点的坐标可通过解如下方程得到：

$$t^2+(s\cdot t-1)^2=1$$

将上式转换一下可得到 $(s^2+1)\cdot t^2=2st$，它的非零解是 $t_1=\frac{2s}{s^2+1}$，于是该交点坐标 $(x,y)$ 有 $x=t_1$，且 $y=s\cdot t_1-1=\frac{s^2-1}{s^2+1}$。显然两个坐标值都是正实数，当且仅当 $s$ 是大于 1 的实数。故当 $s=\frac{m}{n},\ m>n>0$ 时，$x=\frac{2mn}{m^2+n^2},\ y=\frac{m^2-n^2}{m^2+n^2}$，这时我们就可得到勾股数组 $(a,b,c)=(2mn,m^2-n^2,m^2+n^2)$。

假定现有朴素勾股数组 $(a,b,c)$，经过点 $(0,-1)$ 和 $(\frac{a}{c},\frac{b}{c})$ 的直线的斜率是 $\frac{b+c}{a}=\frac{m}{n}$，$\frac{m}{n}$ 是等号左边的化简形式，即是说 $\gcd(m,n)=1$。于是 $T=(2mn,m^2-n^2,m^2+n^2)$ 的各项是 $(a,b,c)$ 各项的倍数。

若 $m,n$ 的奇偶性不同，则 $T$ 也是朴素的，即有 $T=(a,b,c)$。

若 $m,n$ 都是奇数，设 $u=\frac{m+n}{2},\ v=\frac{m-n}{2}$，则，$m=u+v,\ n=u-v$，那么 $u,v$ 中一个是奇数，一个是偶数，则 $\gcd(u,v)=1$。另外：

$$\begin{aligned}
2mn &= 2(u+v)(u-v)=2(u^2-v^2)\\
&\Rightarrow\ mn=u^2-v^2\\
m^2-n^2 &= (u^2+2uv+v^2)-(u^2-2uv+v^2)=4uv\\
&\Rightarrow\ \frac{m^2-n^2}{2}=2uv\\
m^2+n^2 &= (u^2+2uv+v^2)+(u^2-2uv+v^2)=2(u^2+v^2)\\
&\Rightarrow\ \frac{m^2+n^2}{2}=u^2+v^2
\end{aligned}$$

所以 $T_1=(u^2-v^2,2uv,u^2+v^2)$ 是一个朴素勾股数组，它的各项又是 $(a,b,c)$ 各项的倍数，故 $T_1=(a,b,c)$。

# Chapter 10

# 求不超过 N 的所有质数的和

## 10.1 一般的算法

如果你已经有一个相当快速的质数检测算法，像是对问题 7 的分析中给出的那个，你可以直接遍历所有可能是质数的自然数（例如单数或不能被 2 或 3 整除的数），然后将所有检测到的质数累加起来。代码如下：

```
001  limit := 2000000
002  sum := 5    //已知 2 和 3 是质数
003  n := 5
004  while n <= limit
005      if isPrime(n) then sum := sum + n
006      n := n + 2
007      if n <= limit and isPrime(n) then sum := sum + n
008      n := n + 4
009  end while
010  output sum
```

假如 $limit$ 是在 500,0000 以内的话，那么答案大概在一两秒之内就出现了。

虽然说那个质数检测算法（$isPrime$）已经过大量的优化，比检查所有的数字或只仅检查所有的奇数的算法（那些不会在 $\sqrt{n}$ 处停止检测的算法）都要好，但要检测大量的数字的话，这个算法仍然太慢了。要找到所有不超过 100,0000 的质数，若只检测不能被 2 或 3 整除的数字，则要进行多于 2 亿次除法运算，要找到不超过 500,0000 的，就要进行超过 20 亿次除法运算。

这么巨大的工作量实在太可怕了，怎么才能在根本上减少运算量呢?

除了一些更高级的算法，对于这个问题最好的回答已经活跃了 2000 多年了。当你想找不超过某个数的所有质数时，使用 Eratosthenes 筛选法。当你仅仅想是检测一些不太大的数字时，用试除法就够了。然而，假如你是想检查在 $10^9$ 到 $10^{10}$ 之间的数字时，综合使用这两种方法是非常好的，首先找到不大于 $10^5$ 的所有质数，然后用这些质数来对要检验的数进行试除。

## 10.2 Eratosthenes 筛选法

这个古老的方法的基本原理就是：不直接看每个要检测的数因数是否含

[0]Author: daniel.is.fischer

有非 1 和它本身的因数 $d$，而是标记所有包括因数 $d$ 的合数。既然每个合数都含有质因数，那么只要标记所有是质因数的倍数的合数就行了。经典的算法如下：

1. 列出从 2 到 $N$ 的所有数字。
2. 寻找下一个没有被勾掉的数字 $p$，显然 $p$ 是质数。若 $p$ 大于 $\sqrt{N}$，跳到第 5 步。
3. 勾掉除 $p$ 之外所有是 $p$ 的倍数的数字。
4. 跳到第 2 步。
5. 剩下的没被勾掉的数字就是不超过 $N$ 的所有质数。

你只需要从 $p^2$ 开始勾掉 $p$ 的倍数，因为比 $p^2$ 更小的 $p$ 的倍数都已经因含有小于 $p$ 的质因数被勾掉了。这也是我们能够在达到 $\sqrt{N}$ 的时候停止检测的原因。

### 10.2.1 第一个应用实例

让我们从一个实现了类似上面描述的算法的程序开始吧，一会讨论过后我们还会再改进这个程序。对于生成的数列，我们再用一个布尔数组（数组各元素的的值只有真或假）来作为“筛子”，表示哪些数字已经被勾掉。代码如下：

```
001  limit := 2000000
002  crosslimit := floor(sqrt(limit))
003   sieve := new boolean array [2 .. limit] false // 各元素初始值为 false
004  for n := 4 to limit with step 2     // n 每次循环递增 2
005      sieve[n] := true
006  end for
007  for n := 3 to crosslimit with step 2
008      if not sieve[n] then    // 若 n 没被标记，则是质数
009          for m := n*n to limit with step 2*n
010              sieve[m] := true
011          end for
012      end if
013  end for
014  sum := 0
015  for n := 2 to limit
016      if not sieve[n] then
017          sum := sum + n
018      end if
019  end for
020  output sum
```

首先我们把布尔数组的各元素初始化为 false 来表示还没有勾掉任何的数字。在勾掉数字时，我们分开奇数和偶数，稍微和上面描述的古老算法有点不同。在勾掉所有大于 2 的偶数之后，已经没有必要再用它们的奇因数来再次勾掉它们了，因此，当后面要勾掉所有大于奇质数 $p$ 的倍数时，我们可以每次递增 $2p$ 而不

是 $p$，以免重新勾掉偶数。正如上面提到过的，所有小于 $p^2$ 的 $p$ 的倍数已经因含有小于 $p$ 的质因数被勾掉了，因此我们不必再访问这些数字。另外，如果 $n$ 是一个奇合数，当循环到访问 $n$ 时，这个数和它的倍数都早已被勾掉了，因此不必再进行内部循环了。

### 10.2.2 优化这个筛子

一个最容易的优化手段就是根据区分奇数和偶数提出的。除了 2 之外，其余偶数都会被直接勾掉还要占据空间。如果我们扔掉所有偶数，就能省去记录 $\frac{limit-2}{2}$ 个数字的是否被勾掉的信息，更重要的是，还能省掉一半的内存空间。这样就允许我们增大筛子的大小并减少内存浪费，于是筛子的性能就增强了。

只筛选奇数（不包括 1）的话就要对布尔数组的下标索引进行小小的改造。我们要让布尔数组的第 $i$ 个元素指向奇数 $2i+1$，于是，假如 $p = 2i+1$，那么 $p^2 = 4i^2+4i+1$ 对应的布尔数组的下标则是 $2i(i+1)$，并且假如 $m = k \cdot p$ 对应下标 $j$ 的话，那么 $m+2p$ 对应的下标则是 $j+p$。因此，如果第 $i$ 个奇数没被勾掉的话，内部循环就要从第 $2i(i+1)$ 开始，且每下一步循环处理之后的第 $2i+1$ 个奇数。

优化后的代码如下：

```
001  limit = 2000000
002  sievebound := (limit - 1) div 2     // 筛子最后一个下标值
003  sieve := new boolean array [1 .. sievebound] false
004  crosslimit := floor(sqrt(limit) - 1) div 2
005  for i := 1 to crosslimit
006      if not sieve[i] then    // 若 2*i+1 是质数则标记大于它的倍数
007          for j := 2*i*(i+1) to sievebound with step 2*i+1
008              sieve[j] := true
009          end for
010      end if
011  end for
012  sum := 2    // 2 是质数
013  for i := 1 to sievebound
014      if not sieve[i] then sum := sum + (2*i+1)
015  end for
016  output sum
```

结果不到 1 秒就出来了。