# Find Nth Prime

## 实验目的

1. 夯实C/C++基本功
2. 掌握快速找素数的方法（筛法）
3. 锻炼学习优化方法
4. 提升debug能力

## 实验内容

1. 快速找到第n个素数，n为1-100000000的整数

## 实验算法

1. 使用筛法，不断筛除合数，留下素数

## 算法

1. 判定n是否为素数，只需要√n之前的素数即可
2. 第一亿个素数是2038074743，那么只需要√2038074743之前的素数就够了。幸运的是，√2038074743是45145，也就是说，只需要存储45145之前的素数即可。
3. 将45145之前素数存储在内存中，从45145之后得素数不需要存储。当需要45145前的素数时，可直接从数组中取。
4. 利用45145之前的素数对之后得自然数分段筛除，我选取的段大小为60000。素数标为true，合数标为false。筛除完之后，从数组开头向后摸，遇到true则素数数增加1，直到为n。

## 问题和解决方法

Q：经查找，第1亿个素数是2038074743.筛法不可能一次性开辟这么大的内存

A：采用局部筛的方法，一段一段的筛除。这样内存就够使用了。而且用bool数组比int数组占用内存更小。1个Bool类型字符只占用1字节，而int型占用4字节。当然，如果用bit位的0和1来表示是否为合数占用内存更小。不过没时间再去改了。除此之外，数组大小开辟的并非越大越好。计算机存取速度最快的地方在L1，所以我们开辟的数组如果能在L1中运算的话则会是最快的。意识到这一点之后调整数组大小，使得时间开销最低。

Q：会出现一个合数被重复筛除的现象，这导致了时间开销的增加。

A：我并没有完全消除掉这部分的冗余，只是低成本的降低了一些循环浪费。1.对数组赋初值时，2和3的倍数可以先处理掉赋值为false，那么之后就不需要用素数2和3作为基数来筛除了。事实上2的倍数是合数中最多的，占1/2,如果能省去2的筛除，则能省一半时间开销。如果再省去3的筛除则又减少了1/6的工作量（这里不是1/3，因为一些既是2的倍数又是3的倍数已经被筛掉了）。当然如果能预先处理掉更多小素数的倍数，那么就会节省大量的时间。不过再往后我没有找到正规的算法，不停地分情况讨论简直让程序不能更丑，遂放弃…

## 代码

<https://github.com/Javonmeng/Prime>

## 性能分析

性能和编译器关系蛮大的。Dev-C++和VS\_2015(debug)版找第一亿个素数均需要17s左右，而VS\_2015(release)版本只需要7-8s。