[素数判断算法(高效率）](http://blog.csdn.net/liukehua123/article/details/5482854)

标签： [算法](http://www.csdn.net/tag/%e7%ae%97%e6%b3%95)[优化](http://www.csdn.net/tag/%e4%bc%98%e5%8c%96)[google](http://www.csdn.net/tag/google)[io](http://www.csdn.net/tag/io)[百度](http://www.csdn.net/tag/%e7%99%be%e5%ba%a6)

2010-04-13 21:58 96452人阅读 [评论](http://blog.csdn.net/liukehua123/article/details/5482854#comments)(36) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/liukehua123/article/details/5482854#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

ACM（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

chuanbindeng 的 素数判断算法

关于素数的算法是信息学竞赛和程序设计竞赛中常考的数论知识，在这里我跟大家讲一下寻找一定范围内素数的几个算法。看了以后相信

对大家一定有帮助。

    正如大家都知道的那样，一个数 n 如果是合数，那么它的所有的因子不超过sqrt(n)--n的开方,那么我们可以用这个性质用最直观的方法

来求出小于等于n的所有的素数。

    num = 0;

    for(i=2; i<=n; i++)

    {  for(j=2; j<=sqrt(i); j++)

         if( j%i==0 ) break;

       if( j>sqrt(i) ) prime[num++] = i;  //这个prime[]是int型，跟下面讲的不同。

    }

    这就是最一般的求解n以内素数的算法。复杂度是o(n\*sqrt(n))，如果n很小的话，这种算法（其实这是不是算法我都怀疑，没有水平。当然没

接触过程序竞赛之前我也只会这一种求n以内素数的方法。-\_-~）不会耗时很多.

    但是当n很大的时候，比如n=10000000时，n\*sqrt(n)>30000000000,数量级相当大。在一般的机子它不是一秒钟跑不出结果，它是好几分钟都跑不

出结果，这可不是我瞎掰的，想锻炼耐心的同学不妨试一试~。。。。

    在程序设计竞赛中就必须要设计出一种更好的算法要求能在几秒钟甚至一秒钟之内找出n以内的所有素数。于是就有了素数筛法。

    （我表达得不清楚的话不要骂我，见到我的时候扁我一顿我不说一句话。。。）

    素数筛法是这样的：

    1.开一个大的bool型数组prime[]，大小就是n+1就可以了.先把所有的下标为奇数的标为true,下标为偶数的标为false.

    2.然后：

      for( i=3; i<=sqrt(n); i+=2 )

      {   if(prime[i])

          for( j=i+i; j<=n; j+=i ) prime[j]=false;

      }

    3.最后输出bool数组中的值为true的单元的下标，就是所求的n以内的素数了。

    原理很简单，就是当i是质(素)数的时候，i的所有的倍数必然是合数。如果i已经被判断不是质数了，那么再找到i后面的质数来把这个质

数的倍数筛掉。

    一个简单的筛素数的过程：n=30。

    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

    第 1 步过后2 4 ... 28 30这15个单元被标成false,其余为true。

    第 2 步开始：

     i=3;  由于prime[3]=true, 把prime[6], [9], [12], [15], [18], [21], [24], [27], [30]标为false.

     i=4;  由于prime[4]=false,不在继续筛法步骤。

     i=5;  由于prime[5]=true, 把prime[10],[15],[20],[25],[30]标为false.

     i=6>sqrt(30)算法结束。

    第 3 步把prime[]值为true的下标输出来：

     for(i=2; i<=30; i++)

     if(prime[i]) printf("%d ",i);

    结果是 2 3 5 7 11 13 17 19 23 29

    这就是最简单的素数筛选法，对于前面提到的10000000内的素数，用这个筛选法可以大大的降低时间复杂度。把一个只见黑屏的算法

优化到立竿见影，一下就得到结果。关于这个算法的时间复杂度，我不会描述，没看到过类似的记载。只知道算法书上如是说：前几年比

较好的算法的复杂度为o(n),空间复杂度为o(n^(1/2)/logn).另外还有时间复杂度为o(n/logn),但空间复杂度为O(n/(lognloglogn))的算法。

我水平有限啦，自己分析不来。最有说服力的就是自己上机试一试。下面给出这两个算法的程序：

//最普通的方法：

#include<stdio.h>

#include<math.h>

#define N 10000001

int prime[N];

int main()

{

    int i, j, num = 0;

 for(i=2; i<N; i++)

    {  for(j=2; j<=sqrt(i); j++)

         if( j%i==0 ) break;

       if( j>sqrt(i) ) prime[num++] = i;

    }

 for(i=2; i<100; i++) //由于输出将占用太多io时间，所以只输出2-100内的素数。可以把100改为N

    if( prime[i] )printf("%d ",i);

 return 0;

}

//用了筛法的方法：

#include<stdio.h>

#include<math.h>

#define N 10000001

bool prime[N];

int main()

{

   int i, j;

   for(i=2; i<N; i++)

  if(i%2) prime[i]=true;

  else prime[i]=false;

   for(i=3; i<=sqrt(N); i++)

   {   if(prime[i])

       for(j=i+i; j<N; j+=i) prime[i]=false;

   }

   for(i=2; i<100; i++)//由于输出将占用太多io时间，所以只输出2-100内的素数。可以把100改为N

    if( prime[i] )printf("%d ",i);

   return 0;

}

装了vc的同学上机跑一下这两个程序试一试。这个差别，绝对是天上地下。前面那个程序绝对是n分钟黑屏的说。

另外，对于这样的筛法，还可以进一步优化，就是bool型数组里面只存奇数不存偶数。如定义prime[N],则0表示

3，1表示5，2表示7，3表示9...。如果prime[0]为true,则表示3时素数。prime[3]为false意味着9是合数。

这样的优化不是简单的减少了一半的循环时间，比如按照原始的筛法，数组的下标就对应数。则在计算30以内素

数的时候3个步骤加起来走了15个单位时间。但是用这样的优化则是这样：

则由于只存3 5 7 9 11 13 15 17 19 21 23 25 27 29，只需要14个单元

第 1 步 把14个单元赋为true (每个单元代表的数是2\*i+3,如第0单元代表3，第1单元代表5...)

第 2 步开始：

     i=0;  由于prime[0]=true, 把 [3], [6], [9], [12]标为false.

     i=1;  由于prime[1]=true, 把 [6], [11]标为false

     i=2  2\*i+3>sqrt(30)算法结束。

这样优化以后总共只走6个单位时间。

当n相当大以后这样的优化效果就更加明显，效率绝对不仅仅是翻倍。

出了这样的优化以外，另外在每一次用当前已得出的素数筛选后面的数的时候可以一步跳到已经被判定不是素数的

数后面，这样就减少了大量的重复计算。（比如我们看到的，i=0与i=1时都标了[6],这个就是重复的计算。）

我们可以发现一个规律，那就是3（即i=0）是从下标为[3]的开始筛的，5（即i=1）是从下标为[11]开始筛的（因为[6]

已经被3筛过了）。然后如果n很大的话，继续筛。7（i=2）本来应该从下标为[9]开始筛，但是由于[9]被筛过了，而

[16]也已经被5（i=1）筛过了。于是7（i=2）从[23]（就是2\*23+3=49）开始筛。

于是外围循环为i时，内存循环的筛法是从 i+(2\*i+3)\*(i+1)即i\*(2\*i+6)+3开始筛的。

这个优化也对算法复杂度的降低起到了很大的作用。

相比于一般的筛法，加入这两个优化后的筛法要高效很多。高兴去的同学可以试着自己编写程序看一看效率。我这里

有程序，需要的可以向我要。不懂得也可以问我。

上面的素数筛法是所有程序设计竞赛队员都必须掌握的，而后面加了两个优化的筛法是效率很高的算法，是湖南大学

huicpc39同学设计的（可能是学来的，也可能是自创的。相当强悍）。在数量级更大的情况下就可以发现一般筛法和

优化后的筛法的明显区别。

另外，台湾的ACMTino同学也给我介绍了他的算法：a是素数，则下一个起点是a\*a,把后面的所有的a\*a+2\*i\*a筛掉。

这上面的所有的素数筛选的算法都可以再进一步化为二次筛选法，就是欲求n以内的素数，就先把sqrt(n)内的素数求

出来，用已经求得的素数来筛出后面的合数。

我把一般的筛选法的过程详细的叙述了一遍，应该都懂了吧？后面的优化过程及不同的方法，能看懂最好。不是很难的。

相关知识：

最大公约数只有1和它本身的数叫做质数（素数）——这个应该知道吧？-\_-b

    至今为止，没有任何人发现素数的分布规律，也没有人能用一个公式计算出所有的素数。关于素数的很多的有趣的性质或者科学家的努力

我不在这里多说，大家有兴趣的话可以到百度或google搜一下。我在下面列出了一个网址，上面只有个大概。更多的知识需要大家一点一点

地动手收集。

http://www.scitom.com.cn/discovery/universe/home01.html

1.高斯猜测，n以内的素数个数大约与n/ln(n)相当，或者说，当n很大时，两者数量级相同。这就是著名的素数定理。

2.十七世纪费马猜测，2的2^n次方+1，n=0，1，2…时是素数，这样的数叫费马素数，可惜当n=5时，2^32+1就不是素数，

  至今也没有找到第六个费马素数。

3.18世纪发现的最大素数是2^31-1，19世纪发现的最大素数是2^127-1，20世纪末人类已知的最大素数是2^859433-1，用十进制表示，这是一个258715位的数字。

4.孪生素数猜想：差为2的素数有无穷多对。目前知道的最大的孪生素数是1159142985×2^2304－1和1159142985×2^2304＋1。

5.歌德巴赫猜想：大于2的所有偶数均是两个素数的和，大于5的所有奇数均是三个素数之和。其中第二个猜想是第一个的自然推论，因此歌德巴赫猜想又被称为1+1问题。我国数学家陈景润证明了1+2，即所有大于2的偶数都是一个素数和只有两个素数因数的合数的和。国际上称为陈氏定理。