# MILLER 素数检测

<http://www.matrix67.com/blog/archives/234>

poj 1811

不满足2^(n-1) mod n = 1的n一定不是素数；如果满足的话则多半是素数。

 但我们可以考虑a=2，a=3，a=5的情况。对于式子a^(n-1) mod n，取不同的a可能导致不同的结果。一个合数可能在a=2时通过了测试，但a=3时的计算结果却排除了素数的可能。于是，人们扩展了伪素数的定义，称满足a^(n-1) mod n = 1的合数n叫做以a为底的伪素数(pseudoprime to base a)。前10亿个自然数中同时以2和3为底的伪素数只有1272个，这个数目不到刚才的1/4。这告诉我们如果同时验证a=2和a=3两种情况，算法出错的概率降到了0.000025。容易想到，选择用来测试的a越多，算法越准确。通常我们的做法是，随机选择若干个小于待测数的正整数作为底数a进行若干次测试，只要有一次没有通过测试就立即把  
这个数扔回合数的世界。这就是Fermat素性测试。  
    人们自然会想，如果考虑了所有小于n的底数a，出错的概率是否就可以降到0呢？没想到的是，居然就有这样的合数，它可以通过所有a的测试（这个说法不准确，详见我在地核楼层的回复）。Carmichael第一个发现这样极端的伪素数，他把它们称作Carmichael数。你一定会以为这样的数一定很大。错。第一个Carmichael数小得惊人，仅仅是一个三位数，561。前10亿个自然数中Carmichael数也有600个之多。Carmichael数的存在说明，我们还需要继续加强素性判断的算法。

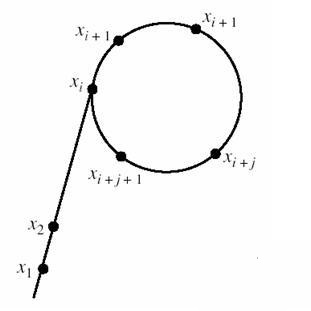
新的测试基于下面的定理：如果p是素数，x是小于p的正整数，且x^2 mod p = 1，那么要么x=1，要么x=p-1。这是显然的，因为x^2 mod p = 1相当于p能整除x^2-1，也即p能整除(x+1)(x-1)。由于p是素数，那么只可能是x-1能被p整除(此时x=1)或x+1能被p整除(此时x=p-1)。  
    我们下面来演示一下上面的定理如何应用在Fermat素性测试上。前面说过341可以通过以2为底的Fermat测试，因为2^340 mod 341=1。如果341真是素数的话，那么2^170 mod 341只可能是1或340；当算得2^170 mod 341确实等于1时，我们可以继续查看2^85除以341的结果。我们发现，2^85 mod 341=32，这一结果摘掉了341头上的素数皇冠，面具后面真实的嘴脸显现了出来，想假扮素数和我的素MM交往的企图暴露了出来。  
    这就是Miller-Rabin素性测试的方法。不断地提取指数n-1中的因子2，把n-1表示成d\*2^r（其中d是一个奇数）。那么我们需要计算的东西就变成了a的d\*2^r次方除以n的余数。于是，a^(d \* 2^(r-1))要么等于1，要么等于n-1。如果a^(d \* 2^(r-1))等于1，定理继续适用于a^(d \* 2^(r-2))，这样不断开方开下去，直到对于某个i满足a^(d \* 2^i) mod n = n-1或者最后指数中的2用完了得到的a^d mod n=1或n-1。这样，Fermat小定理加强为如下形式：  
    尽可能提取因子2，把n-1表示成d\*2^r，如果n是一个素数，那么或者a^d mod n=1，或者存在某个i使得a^(d\*2^i) mod n=n-1 ( 0<=i<r ) （注意i可以等于0，这就把a^d mod n=n-1的情况统一到后面去了）  
    Miller-Rabin素性测试同样是不确定算法，我们把可以通过以a为底的Miller-Rabin测试的合数称作以a为底的强伪素数(strong pseudoprime)。第一个以2为底的强伪素数为2047。第一个以2和3为底的强伪素数则大到1 373 653。  
    Miller-Rabin算法的代码也非常简单：计算d和r的值（可以用位运算加速），然后二分计算a^d mod n的值，最后把它平方r次。

对于大数的素性判断，目前Miller-Rabin算法应用最广泛。一般底数仍然是随机选取，但当待测数不太大时，选择测试底数就有一些技巧了。比如，如果被测数小于4 759 123 141，那么只需要测试三个底数2, 7和61就足够了。当然，你测试的越多，正确的范围肯定也越大。如果你每次都用前7个素数(2, 3, 5, 7, 11, 13和17)进行测试，所有不超过341 550 071 728 320的数都是正确的。如果选用2, 3, 7, 61和24251作为底数，那么10^16内唯一的强伪素数为46 856 248 255 981。这样的一些结论使得Miller-Rabin算法在OI中非常实用。通常认为，Miller-Rabin素性测试的正确率可以令人接受，随机选取k个底数进行测试算法的失误率大概为4^(-k)。

Pollard\_rho**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**的大致流程是 先判断当前数是否是素数（Miller\_rabin）了，如果是则直接返回。如果不是素数的话，试图找到当前数的一个因子（可以不是质因子）。然后递归对该因子和约去这个因子的另一个因子进行分解。

那么自然的疑问就是，怎么找到当前数n的一个因子？当然不是一个一个慢慢试验，而是一种神奇的想法。其实这个找因子的过程我理解的不是非常透彻，感觉还是有一点儿试的意味，但不是盲目的枚举，而是一种随机化算法。我们假设要找的因子为p，他是随机取一个x1，由x1构造x2，使得｛p可以整除x1-x2 && x1-x2不能整除n｝则p=gcd（x1-x2，n），结果可能是1也可能不是1。如果不是1就找寻成功了一个因子，返回因子；如果是1就寻找失败，那么我们就要不断调整x2，具体的办法通常是x2=x2\*x2+c（c是自己定的）直到出现x2出现了循环==x1了表示x1选取失败重新选取x1重复上述过程。（似乎还存在一个每次找寻范围\*2的优化，但是不太懂。。。）

因为x1和x2再调整时最终一定会出现循环，形成一个类似希腊字母rho的形状，故因此得名。



另外通过find函数来分解素数，如果找到了一个素数因子则加入到因子map中，否则如果用Pollard找到一个因子则递归去找素数因子