added: 16 Jan 2009 0:58
Edited: 20 Aug 2012 23:51

Primitive roots

Contents [hide]

Definition

A primitive root modulo n(Primitive root modulo n) is called a number $g$that all his power modulo $n$run through all the numbers relatively prime to n.
Mathematically, it is formulated as follows: if $g$is a primitive root modulo n, then for any integer a such that $\gcd(a, n) = 1$, there exists an integer such $k$that $g^k \equiv a \pmod n$.

In particular, for the case of the simple $n$power of a primitive root run through all the numbers from 1 before $n - 1$.

The existence of

Primitive root modulo $n$if and only if there $n$is a degree of odd prime or twice a prime power, and also in cases $n = 1, n = 2, n = 4$.

This theorem (which was fully proved by Gauss in 1801) is given here without proof.

Communication with the function of the Euler

Let g- a primitive root modulo n. Then we can show that the smallest number $k$for which $g^k \equiv 1 \pmod n$(ie k- index g(multiplicative order)), power $\phi(n)$. Moreover, the converse is also true, and this fact will be used by us in the following algorithm for finding a primitive root.

Furthermore, if the modulus $n$is at least one primitive root, the total of $\phi(\phi(n))$(because cyclic group with $k$elements has $\phi(k)$generators).

Algorithm for finding a primitive root

A naive algorithm would require for each test value of time to calculate all of its power and verify that they are all different. It's too slow algorithm, below we are using several well-known theorems of number theory, we obtain a faster algorithm.$g O(n)$

Above we present a theorem that if the smallest number $k$for which $g^k \equiv 1 \pmod n$(ie k- index g) as well $\phi(n)$, then g- a primitive root. Since for any number $a$relatively prime to n, performed Euler's theorem ($a^{\phi(n)} \equiv 1 \pmod n$), then to check that the $g$primitive root, it suffices to verify that for all numbers $d$smaller $\phi(n)$, satisfied $g^d \not\equiv 1 \pmod n$. However, while it is too slow algorithm.

From Lagrange's theorem that the rate of any number modulo na divisor $\phi(n)$. Thus, it suffices to show that for all proper divisors $d \mid \phi(n)$is performed $g^d \not\equiv 1 \pmod n$. This is a much faster algorithm, but we can go further.

We factor the number $\phi(n) = p_1^{a_1} \dots p_s^{a_s}$. We prove that in the previous algorithm, it suffices to

consider as da number of the form $\frac{\phi(n)}{p_i}$. Indeed, suppose that d- any proper subgroup $\phi(n)$. Then, obviously, there exists such $j$that $d \mid \frac{\phi(n)}{p_j}$, ie $d \cdot k = \frac{\phi(n)}{p_j}$. However, if $g^d \equiv 1 \pmod n$we would get:

$$g^{\frac{\phi(n)}{p_j}} \equiv g^{d \cdot k} \equiv \left(g^d\right)^k \equiv 1^k \equiv 1 \pmod n,$$

ie still among the numbers of the form $\frac{\phi(n)}{p_i}$there would be something for which the condition is not satisfied, as required.

Thus, an algorithm for finding a primitive root. Find $\phi(n)$, factorize it. Now iterate through all the numbers $g = 1 \dots n$, and for each consider all the values $g^{\frac{\phi(n)}{p_i}} \pmod n$. If the current $g$all these numbers were different from 1, this $g$is the desired primitive root.

Time of the algorithm (assuming that the number $\phi(n)$has $O\left(\log \phi(n)\right)$divisors, and exponentiation algorithm performed binary exponentiation , ie, for $O(\log n)$) power $O\left(\text{Ans} \cdot \log \phi(n) \cdot \log n\right)$, plus the time of the factorization $\phi(n)$, where Ans- the result, ie value of the unknown primitive root.

About the growth rate of the growth of primitive roots $n$are known only estimates. It is known that primitive roots - a relatively small amount. One of the famous assessment - assessment Shupe (Shoup), that, assuming the truth of the Riemann hypothesis, there is a primitive root $O(\log^6 n)$.

Implementation

Function powmod () performs a binary exponentiation modulo a function generator (int p) - is a primitive root modulo a prime p(factorization of $\phi(n)$the simplest algorithm is implemented for $O(\sqrt{\phi(n)})$).

To adapt this function to arbitrary p, just add the calculation of Euler's function in a variable phi, as well as weed out resnon-prime to n.

```
int powmod (int a, int b, int p) {
        int res = 1;
        while (b)
                if (b & 1)
                        res = int (res * 1ll * a % p),  --b;
                else
                        a = int (a * 1ll * a % p),  b >>= 1;
        return res;
}

int generator (int p) {
        vector<int> fact;
        int phi = p-1,  n = phi;
        for (int i=2; i*i<=n; ++i)
                if (n % i == 0) {
                        fact.push_back (i);
                        while (n % i == 0)
                                n /= i;
                }
        if (n > 1)
                fact.push_back (n);

        for (int res=2; res<=p; ++res) {
                bool ok = true;
                for (size_t i=0; i<fact.size() && ok; ++i)
                        ok &= powmod (res, phi / fact[i], p) != 1;
```

```
                if (ok)  return res;
        }
        return -1;
}
```

5 Комментариев e-maxx Войти

Лучшее вначале Поделиться Избранный

Присоединиться к обсуждению...

ksjuchi • 2 года назад

В случае произвольного p не требуется ли ввести дополнительную проверку на взаимопростоту g и n при переборе возможных g? Ведь теорема Эйлера выполняется только для взаимнопростых чисел.

• Ответить • Поделиться ›

e_maxx Модератор ksjuchi • 2 года назад

Да, вы правы.

• Ответить • Поделиться ›

Ivan Nikulin e_maxx • 2 года назад

Кстати, если p=2*k^i, где k - простое, то нужно еще проверять в конце, будет ли p^phi=1.

• Ответить • Поделиться ›

Ivan Nikulin • 2 года назад

Зачем res<=p вместо res<p? Просто="" очевидно="" же,="" что="" p="" не="" является="" первообразным="" корнем="" p="" :)="">

• Ответить • Поделиться ›

e_maxx Модератор Ivan Nikulin • 2 года назад

Ну да, можно и строго меньше поставить.

• Ответить • Поделиться ›