Definition

Primitive root modulo n(Primitive root modulo n) called a number $g$that all its powers modulo $n$run through all the numbers relatively prime to n. Mathematically, this 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 1before $n - 1$.

Existence

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

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

Communication with the Euler function

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 later in our algorithm for finding primitive roots.

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

Algorithm for finding a primitive root

Naive algorithm requires values for each test time to calculate all his power and verify that they are all different. It's too slow algorithm, below we are using several well-known theorems of number theory 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 verify that the $g$primitive root, it suffices to verify that for all numbers $d$less than$\phi(n)$operating $g^d \not\equiv 1 \pmod{n}$. However, until it is too slow algorithm.

From Lagrange's theorem that the index of any number modulo n a divisor $\phi(n)$. Thus, it suffices to verify 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.

Factorize number $\phi(n) = p_1^{a_1} \cdots p_s^{a_s}$. We prove that in the previous algorithm, it suffices to consider as d a number of the form $\frac{\phi(n)}{p_i}$. Indeed, let d - any proper subgroup $\phi(n)$. Then, obviously, there is 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 kind of numbers $\frac{\phi(n)}{p_i}$ there would be something for which the condition is not met, as required.

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

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

About the rate of growth with the growth of primitive roots n are known only estimates. It is known that primitive roots - relatively small quantities. One of the famous guest - evaluation 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 (number factorization $\phi(n)$ is carried out for a simple algorithm $O(\sqrt{\phi(n)})$).

To adapt this function to arbitrary p, just add the calculation of Euler's function in a variable phi, and discarding of res non-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;
}
```