CP-Algorithms

Page Authors

Search

Discrete Root

Table of Contents

The problem of finding discrete root is defined as follows. Given a prime n and two integers a and k, find all x for which:

$$x^k \equiv a \pmod n$$

The algorithm

We will solve this problem by reducing it to the discrete logarithm problem.

Let's apply the concept of a primitive root modulo n. Let g be a primitive root modulo n. Note that since n is prime, it must exist, and it can be found in $O(Ans \cdot \log \phi (n) \cdot \log n) = O(Ans \cdot \log^2 n)$ plus time of factoring $\phi (n)$.

We can easily discard the case where $a = 0$. In this case, obviously there is only one answer: $x = 0$.

Since we jnow that n is a prime, any number between 1 and $n-1$ can be represented as a power of the primitive root, and we can represent the discrete root problem as follows:

$$(g^y)^k \equiv a \pmod n$$

where

$$x \equiv g^y \pmod n$$

This, in turn, can be rewritten as

$$(g^k)^y \equiv a \pmod n$$

Now we have one unknown y, which is a discrete logarithm problem. The solution can be found using Shanks' baby-step-giant-step algorithm in $O(\sqrt{n} \log n)$ (or we can verify that there are no solutions).

Having found one solution y_0, one of solutions of discrete root problem will be $x_0 = g^{y_0} \pmod n$.

Finding all solutions from one known solution

To solve the given problem in full, we need to find all solutions knowing one of them: $x_0 = g^{y_0} \pmod n$.

Let's recall the fact that a primitive root always has order of $\phi(n)$, i.e. the smallest power of g which gives 1 is $\phi(n)$. Therefore, if we add the term $\phi(n)$ to the exponential, we still get the same value:

$$x^k \equiv g^{y_0 \cdot k + l \cdot \phi(n)} \equiv a \pmod n \forall l \in Z$$

Hence, all the solutions are of the form:

$$x = g^{y_0 + \frac{l \cdot \phi(n)}{k}} \pmod n \forall l \in Z.$$

where l is chosen such that the fraction must be an integer. For this to be true, the numerator has to be divisible by the least common multiple of $\phi(n)$ and k. Remember that least common multiple of two numbers $lcm(a, b) = \frac{a \cdot b}{gcd(a,b)}$; we'll get

$$x = g^{y_0 + i \frac{\phi(n)}{gcd(k, \phi(n))}} \pmod n \forall i \in Z.$$

This is the final formula for all solutions of discrete root problem.

Implementation

Here is a full implementation, including routines for finding the primitive root, discrete log and finding and printing all solutions.

```
int gcd (int a, int b) {
    return a ? gcd (b%a, a) : b;
}

int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --
        else
            a = int (a * 1ll * a % p),  b >>=
    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;
            ok &= powmod (res, phi / fact[i],
        if (ok)  return res;
    }
    return -1;
}

int main() {

    int n, k, a;
    cin >> n >> k >> a;
    if (a == 0) {
        puts ("1\n0");
        return 0;
    }

    int g = generator (n);

    int sq = (int) sqrt (n + .0) + 1;
    vector < pair<int,int> > dec (sq);
    for (int i=1; i<=sq; ++i)
        dec[i-1] = make_pair (powmod (g, int (
    sort (dec.begin(), dec.end());
    int any_ans = -1;
    for (int i=0; i<sq; ++i) {
```

```
            int my = int (powmod (g, int (i * 1ll
            vector < pair<int,int> >::iterator it
                lower_bound (dec.begin(), dec.end(
            if (it != dec.end() && it->first == my
                any_ans = it->second * sq - i;
                break;
            }
        }
        if (any_ans == -1) {
            puts ("0");
            return 0;
        }

        int delta = (n-1) / gcd (k, n-1);
        vector<int> ans;
        for (int cur=any_ans%delta; cur<n-1; cur+=
            ans.push_back (powmod (g, cur, n));
        sort (ans.begin(), ans.end());
        printf ("%d\n", ans.size());
        for (size_t i=0; i<ans.size(); ++i)
            printf ("%d ", ans[i]);

    }
```