Discrete root extract

Discrete task of extracting the root (similar to the discrete logarithm problem) is as follows. According to n(n- idle) a, $k$you want to find all $x$satisfying:

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

An algorithm for solving

Will solve the problem by reducing it to the discrete logarithm problem.

To do this, apply the concept of primitive roots modulo$n$. Let g- a primitive root modulo n(because n- simple, it exists). We can find it, as described in the related article, for $O(\mathrm{Ans} \cdot \log \phi(n) \cdot \log n) = O(\mathrm{Ans} \cdot \log^2 n)$the time plus the number factorization $\phi(n)$.

Immediately discard the case when $a = 0$- in this case we immediately find the answer $x = 0$.

Since in this case (n- prime) from any number $1$to $n-1$be represented in the form of a power of a primitive root, the root of the discrete problem, we can be written as

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

where

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

Trivial transformation we obtain:

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

Here is the unknown quantity y, so we came to the discrete logarithm problem in pure form. This problem can be solved by the algorithm baby-step-giant-step Shanks for $O(\sqrt{n} \log n)$, ie find one of the solutions $y_0$of this equation (or find that this equation has no solution).

Suppose we have found a solution to $y_0$this equation, then one of the solutions of the discrete root will $x_0 = g^{y_0} \pmod n$.

Finding all solutions, knowing one of them

To completely solve the problem, we must learn one found $x_0 = g^{y_0} \pmod n$ find all other solutions.

For this, we recall a fact that a primitive root always has order $\phi(n)$ (see the article on the primitive root), ie the least degree g, giving a unit is $\phi(n)$. Therefore, the addition of the term with the exponent $\phi(n)$ does not change anything:

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

Hence, all the solutions have the form:

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

where l is chosen so that the fraction $\frac{l \cdot \phi(n)}{k}$ was intact. To this fraction was intact, the numerator must be a multiple of the least common multiple $\phi(n)$ and k where (remembering that the least common multiple of two numbers $\mathrm{lcm}(a, b) = \frac{a \cdot b}{\gcd(a,b)}$), we obtain:

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

This is the final convenient formula, which gives a general view of all the solutions of the discrete root.

Implementation

We present a complete implementation, including finding a primitive root, and finding the discrete logarithm and the withdrawal of all decisions.

```
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),  --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;
}

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 (i * sq * 1ll *
k % (n - 1)), n), i);
	sort (dec.begin(), dec.end());
	int any_ans = -1;
	for (int i=0; i<sq; ++i) {
```

```
			int my = int (powmod (g, int (i * 1ll * k % (n - 1)),
n) * 1ll * a % n);
			vector < pair<int,int> >::iterator it =
				lower_bound (dec.begin(), dec.end(), make_pair
(my, 0));
			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+=delta)
			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]);

}
```