Berechnung der Quadratwurzel $x^2 \equiv_n a$ im Fall n ist prim oder $n = pq$ ein Primprodukt

Alexander Weigl
weigla@fh-trier.de

June 8, 2011

1 Einleitung

Die Berechnung der Quadratwurzel von quadratischen Resten ist elementar in manchen kryptologischen Verschlüsselsystemen. Vorliegendes Werk dokumentiert die Erarbeitung einer Software für die Berechnung der Quadratwurzel x zum quadratischen Rest a aus der Gleichung:

$$x^2 \equiv a \mod x \quad \text{mit } \gcd(a, x) = 1 \tag{1}$$

Eine Lösung existiert nicht immer (vgl. Tabelle 1) mit $a = 2$. Sowie $a = 5$ in Z_{15} ist nicht relativ prim zu 15. In diesen Fall bezeichnen wir a als quadratischer Nichrest.

Für die Berechnung des quadratischen Restes fordere ich die Bedingungen

$$n \text{ ist prim} \qquad\qquad n = p \cdot 1 \text{ mit p,q sind prim} \tag{2}$$

Wenn $n = pq$ ein Primprodukt aus zwei Zahlen ist, kann man die Quadratwurzel über die Gleichungen:

$$x^2 = a \mod p \quad x^2 = a \mod p \tag{3}$$

und die Anwendungen des chinesischen Restsatz gelöst werden. Aus (3) ergibt sich auch die Forderungen nach der ersten Bedingung.

Table 1: Beispiel für Quadratwurzel in Z_{15}

a	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	0	1	4	9	1	10	6	4	4	6	10	1	9	4	1

2 Grundlagen

Im Folgenden möchte ich die Berechnung über einen Bottom-Up Ansatz vorstellen.

1. Fall: Sei n prim. Dann kann die Lösung für die Gleichung (1) wie folgt bestimmt werden:

- $a^{(p-1)/4} \mod p$ falls $n \equiv 3 \mod 4$
- Ausprobieren aller $1 \leq i \leq {}^{n}/_{2}$ mit $i^2 = a \mod n$

In diesen Fall können wir überprüfen ob a überhaupt ein quadratischer Rest ist, wenn die Gleichung erfüllt wird:

$$a^{(p-1)/2} \mod n = a \tag{4}$$

2. Fall: Sei $n = pq$ ist ein Primprodukt aus zwei Zahlen. Dann zerlegen wir in n in seine Primfaktoren pq. Dann Lösen wir die Gleichungen (3) und erhalten vier Werte $x'_{pq_{12}}$. Mit den Gleichungen gehen wir in den chinesischen Restsatz:

$$x_1 \equiv x'_{p_1} \mod p \qquad x_1 \equiv x'_{q_1} \mod q \tag{5}$$
$$x_2 \equiv x'_{p_2} \mod p \qquad x_2 \equiv x'_{q_1} \mod q \tag{6}$$
$$x_3 \equiv x'_{p_1} \mod p \qquad x_3 \equiv x'_{q_2} \mod q \tag{7}$$
$$x_4 \equiv x'_{p_2} \mod p \qquad x_4 \equiv x'_{q_2} \mod q \tag{8}$$

und erhalten x_{1234} die unsere vier Lösungen für (1) darstellen.

3 Umsetzung

Der Ablauf für das Finden der Quadratwurzel wurde oben beschrieben. Es fehlen noch die Algorithmen für den Primzahlentest, Primfaktorzerlegung und den chin. Restsatz. Der letztere wurde bereits in anderen Arbeiten für Angewandte Kryptologie beschreiben und wird hier ausgespart.

3.1 Primzahlentest

Der Primzahlentest entscheidet das Problem $p \overset{?}{\in} PRIM$. Für ein $p \in \mathcal{Z}$. Ich verwende dabei für $p <= 10000$ einen Brute-Force-Ansatz indem durch alle Zahlen $2 <= i <= \sqrt{p}$ auf Teilbarkeit prüfen ($\mathcal{O}(2^n)$). Für $p > 10000$ wird ein probalistier Ansatz verwendet,

wenn schon $p \mod 2 = 1$:

```
Wähle a_1, ..., a_k zufällig aus {1, ..., p-1};
foreach a_i do
    t = a_i^((p-1)/2);
    if not t ≡ ±1 mod p then
        p ∉ PRIM;
        break
    end
end
```

Die Fehlerwahrscheinlichkeit sinkt mit größeren k: $\leq 1/2^k$. Die Implementierung ist in Listing 4. Hier wird das k anhand der gg. Fehlertoleranz bestimmt.

3.2 Primfaktorzerlegung

Die Zerlegung von n in die Primfaktoren ist relativ einfach, wenn man die Primzahlen von $1, \ldots, n/2$ kennt. Wir können nun die Teilbarkeit von n und den Primzahlen prüfen. Es gibt ungeführ $\frac{n}{2 \ln n - ln2}$ solcher Primzahlen gibt. Das Finden der Primzahlen erfolgt über das Sieb des Eratosthenes`http://de.wikipedia.org/wiki/Sieb_des_Eratosthenes`:

Algorithmus 1: Finden aller Primzahlen bis n

```
Data : a_1, ... a_n , a_i ∈ {0, 1}
Result : a_i = 1 → i ist prim
Initialisierung a_i = 1;
a_01 = 0;
for i := 2 to n do
    if a_i = 1 then
        for j := 2 · i to n do
            a_j = 0
        end
    end
end
```

Der Algorithmus markiert alle alle Zahlen, die ein Vielfaches einer Primzahl sind. Der Nachteil des Siebes ist ein höher Speichernplatzverbrauch. Die Implementierung des Siebes ist im Listing 1; die des Primfaktorzerlegung in 2.

4 Programmvorstellung

Die Entwicklung des Programmes erfolgte in Java und wurde später mit GWT[1] in Javascript umgewandelt. Die Verwendung von GWT hat den Vorteil der statischen Prüfung der Syntax und Methodensignaturen.

[1] Google Web Toolkit: `http://google.de/?q=GWT`

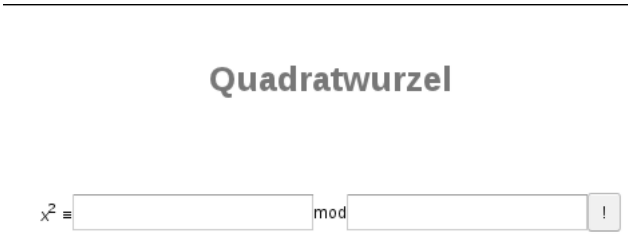

Figure 1: Web-Oberfläche der Applikation

In der Eingabe Maske (Abbildung **??**) kann des a und n in der Form der Gleichung (1) erfolgen. Ergebnisse werden anschließend in der unterhalb dargestellt. Dabei versucht die Applikation alle Berechnung über Proberechnungen zu verfizieren. Sollte es nicht möglich sein die Quadratwurzel zu berechnen wird der Prozess mit einer `NoSolutionException` abgebrochen

5 Fazit

Die Fällen (2) in werden unter der Bedingung das die Zahlen in das `Integer` Datentyp passen erfolgreich berechnet. Das Limit auf den Datentyp `Integer` ruht von der maximalen Größe von Arrays in Java wieder, die beim Sieb des Erosthenes eingesetzt werden.

Im Weiteren gibt es noch einen Sonderfall für (1): bei $p \equiv 1 \mod 4$. Dieser Fall kann in einer späteren Funktion ergänzt werden[2].

Viele Funktionen wie GCD,Primzahlentest oder chin. Restsatz sind implementiert und könnten auch über eine noch zu erstellende Oberfläche für den Benutzer zugänglich gemacht werden.

Source Code: `http://github.com/areku/QuadraticRemainderRoot`

6 Listings

Listing 1: Sieb des Erosthenes

```
1 public static List<Integer> getPrimes(int n) {
2         LinkedList<Integer> primes = new LinkedList<Integer>();
```

[2]Fallunterscheidung in `rootWithPrime`

```
        boolean b[] = new boolean[n];
        for (int i = 2; i < n; i++) {
            if (!b[i]) {
                        primes.add(i);
                        for (int j = i; j < n; j += i) {
                            b[i] = true;
                        }
                }
        }
        return primes;
}
```

Listing 2: Primfaktorzerlegung

```
public static int[] factors(int n) {
        List<Integer> factors = new LinkedList<Integer>();
        List<Integer> primes = getPrimes((int) (n / 2));
        long num = n;

        Iterator<Integer> iter = primes.iterator();
        while (num > 1 && iter.hasNext()) {
            int p = iter.next();
            if (num % p == 0) {
                factors.add(p);
                num = num / p;
            }
        }

        int[] l = new int[factors.size()];
        int i = 0;
        for (int m : factors) {
            l[i++] = m;
        }
        return l;
}
```

Listing 3: GCD

```
 public static int chineseRemainder(int[] a, int m[]) {
        if (a.length != m.length) {
            throw new IllegalArgumentException();
        }

        int n = m.length;

        long product = 1;
        for (int i = 0; i < n; i++) {
            product *= m[i];
        }

        long M[] = new long[n];
        for (int i = 0; i < n; i++) {
            M[i] = product / m[i];
        }
```

```
17
18          long Minv[] = new long[n];
19          for (int i = 0; i < n; i++) {
20              Minv[i] = GCD.extendedGCDFactorFirst(M[i], m[i]);
21          }
22
23          int x = 0;
24          for (int i = 0; i < Minv.length; i++) {
25              x += (a[i] * M[i] * Minv[i]) % product;
26          }
27
28          x%=product;
29
30          if(x<0)
31          {
32              x+=product;
33          }
34          return x;
35  }
```

Listing 4: statistischer Primzahlentest

```
1           public static boolean statTest(int n, double prob) {
2           if ((n & 1) == 0) {
3               return false;
4           }
5
6           if (n <= 10000)// kleiner Bruteforce
7           {
8               for (int i = 3; i < Math.sqrt(n); i++) {
9                   if (n % i == 0)
10                      return false;
11              }
12              return true;
13          }
14
15          int k = 1;
16          double p = 0.5;
17          for (; k < 23; k++) {// Ueberschreitung maschinengenauigkeit
18              if (p <= prob)
19                  break;
20              p /= 2;
21          }
22
23          Random r = new Random();
24          for (int i = 0; i < k; i++) {
25              int ai = r.nextInt((int) n - 1);
26              long t = (long) (Math.pow(ai, (n - 1) / 2)) % n;
27              if (t == 1)
28                  return true;
29          }
30          return false;
31      }
```