Att implementera kvadratiska sållet

Torbjörn Granlund

Avancerade algoritmer 2012

Överblick över algoritmen

Antag att vi ska faktorisera N.

1. Plocka bort småfaktorer med trial division ($+$ ev Pollard ρ)
2. Har vi nu primtal?
3. Identifiera tal av formen a^b där $a, b \geq 2$
4. Bestäm faktorbas-gränsen till $O(e^{\frac{1}{2}\sqrt{\log N \log\log N}})$
5. Räkna fram en faktorbas, dvs primtal med $\left(\frac{N}{p}\right) = 1$
6. Sätt $Q(x) = (\lfloor\sqrt{N}\rfloor + x)^2 - N$
7. Leta efter $Q(x)$ som är glatta över faktorbasen tills vi har fler än $|\mathit{faktorbasen}|$
8. Använd Gauss-elimination över GF(2) över funna $Q(x)$ för att hitta en faktorisering

Plocka bort småfaktorer med trial division (+ ev Pollard ρ)

Trial division kan duga, med inkompilerad primtalstabell.

Antag att vi delat N med alla primtal $p \leq R$.

Har vi nu primtal?

Glöm inte probabilistisk primtalstest!

(QS kan inte faktorisera primtal.)

N en perfekt potens?

Kvadratiska sållet hittar faktorn $a^t | N$, inte $a | N$. Om t ex $N = a^t$ blir det inte så lyckat.

Lösning: Använd GMPs test för perfekta potenser, eller testa direkt att räkna

$$\lfloor \sqrt[k]{N} \rfloor^k \stackrel{?}{=} N$$

för $k = 2, \ldots$

Vi kan begränsa k, beroende på försöksdivisionsgränsen!

Bestäm faktorbas-gränsen till $B = c \cdot e^{\frac{1}{2}\sqrt{\log N \log\log N}}$

Trollformel.

Naturlig logaritm, inte bas-2 som vi datanördar brukar avse.

Konstanten c ska vara ungefär 1. (Min kod har $c = 3$.)

Kalla vald gräns B.

Räkna fram en faktorbas, dvs primtal med $\left(\frac{N}{p}\right) = 1$

Vi ska ha primtal med $p < B$ där N är en kvadratisk rest mod p.

Varför kvadratisk rest? Jo, annars $p \nmid Q(x)$ för varje x.

Algoritm: Brute force eller Euklides-likande. Se Wikipedia-artikeln *Legendre symbol*.

Sätt $Q(x) = (\lfloor \sqrt{N} \rfloor + x)^2 - N$

Polynom valt sådant att $Q(x)$ är en kvadrat mod N.

Leta efter $Q(x)$ som är glatta över faktorbasen

Vi ska ha linjära beroenden, vilket vi ha då vi hittat fler glatta $Q(x)$ än vi har tal i faktorbasen.

Trial division fungerar, men är långsamt.

Notera att om $p^t | Q(x_1)$ så gäller också $p^t | Q(x_1 + p^t)$.
Detta inbjuder till ett såll inte olikt Eratosthenes!

Hur hitta x_1?

Alt 1: Testa $p^t | Q(1)$, $p^t | Q(2)$, $\ldots$, $p^t | Q(p^t)$.

Alt 2: Använd Shanks-Tonelli för att lösa ekv $x^2 \equiv N \pmod{p}$

Not 1: För varje udda p^t har vi **två** lösningar per t.
Not 2: För $p = 2^t$ har 0, 1, 2, eller 4 lösn per t (S-T fungerar ej!).

Glöm inte multipelfaktorer p^t, $t \geq 2$!

Sållning

Så vi har räknat ut p^t och minimala $x : p^t | Q(x)$.
Igen: Flera x per p^t!

Sålla!

Hur ska sålltabellen och sålloperationerna se ut?

1. Sätt element motsvarande x till 1, multiplicera med p
2. Sätt element motsvarande x till $\log Q(x)$, subtrahera $\log(p)$

I alternativ 1 har vi bignumtal i sålltabellen.

När har vi glatthet?

1. De positioner som $= Q(x)$
2. De positioner som ≈ 0

Värden $Q(x)$ som flaggas som glatta behöver faktoriseras, annars vet faktiskt vi inte deras faktorer.

Tidsbesparingstrick i sållning

Använd log-varianten (dvs variant 3 för föreg sida). Använd datatypen `float`. (Det går med heltalstyp, även en enda byte, fast undvik detta innan koden fungerar!)

Jobba i pass med måttligt stor sålltabell. Fuskberäkna $\log Q(x)$ för hela detta intervall till samma värde, t ex för största x i intervallet.

Räkna ut offset för varje p^t för pass 1 på sätt som tidigare beskrivits. Håll sedan reda på offset för varje p^t för nästa sållpass.

Var tolerant mot avrundningsfel, isht som $\log Q(x)$ beräknats för lite fel x.

Använd Gauss-elimination över GF(2) över funna $Q(x)$ för att hitta en faktorisering

Vi behöver inte här hålla redan på exakt multiplicitet av en primfaktor, bara om den har udda (dvs ej är kvadrat) eller jämn (dvs är kvadrat. Dvs exakt GF(2)!

Addition över GF(2) är XOR! 64 GF(2)-variabler per 64-bitsord!

Extremt gles matris kan utnyttjas, rekommenderas ej i version 1!

För exempel, se `http://en.wikipedia.org/wiki/Quadratic_sieve`