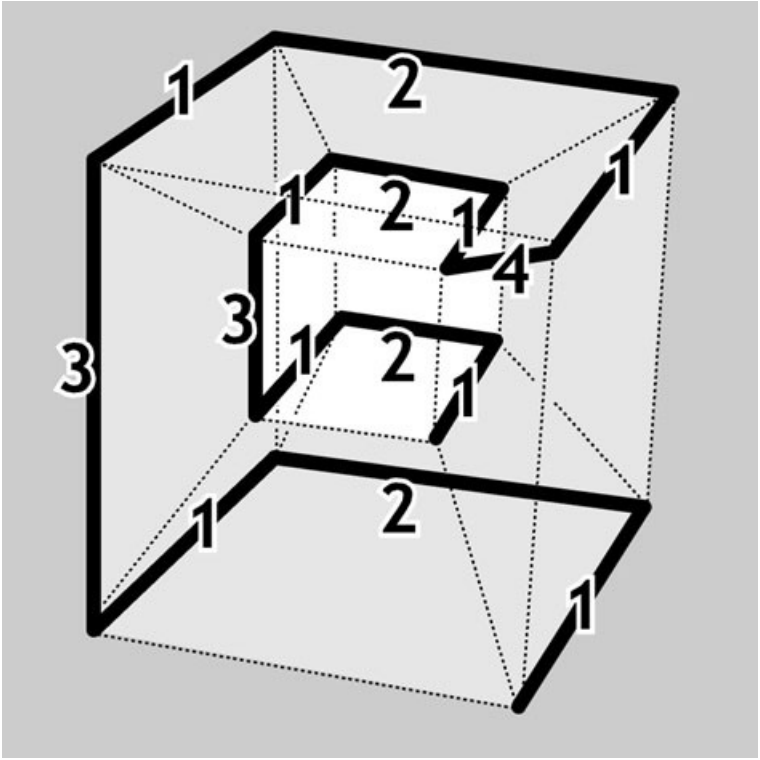
HMPQS für Dummies



Zentralperspektive eines vierdimensionalen Hyperwürfels.
Der Hamiltonsche Pfad erreicht jede der 16 Ecken.

Abstract: HMPQS, Hypercube Polynomial Quadratic Sieve, ist der schnellste bekannte Algorithmus zur Faktorisierung ganzer Zahlen bis zu etwa 110 Dezimalstellen. Für größere Zahlen ist das erheblich kompliziertere GNFS (allgemeines Zahlenkörpersieb) schneller. Diese Beschreibung bezieht sich auf meine Common-LISP Implementierung des „HMPQS with one large prime“ und erklärt die wichtigsten zahlentheoretischen Grundlagen. Bezüge auf den Quelltext sind in der Schrift *abgesetzt. Es folgt zunächst eine Erläuterung des QS, dessen Diskussion das HMPQS motiviert.*

Gesucht ist ein nichttrivialer Teiler von $n \in \mathbb{N}$.

Wenn n zusammengesetzt ist, so wußte schon Fermat, dann läßt es sich als Differenz zweier Quadrate schreiben: $n = (u+v)\cdot(u-v) = u^2 - v^2$. Tatsächlich reicht es mitunter, zwei Quadrate zu finden, die dieser schwächeren Kongruenz genügen:

$$u^2 - v^2 \equiv 0 \ (mod\ n) \ ,$$

denn es folgt direkt, $gcd(u+v, n) \mid n$ und $gcd(u-v, n) \mid n$. Zur Berechnung des größten gemeinsamen Teilers verwendet man z.B. den euklidischen Algorithmus.

Mit $u \equiv \pm v \ (mod\ n)$ würden wir allerdings nur triviale Teiler finden, also suchen wir nach $u^2 \equiv v^2 \ (mod\ n)$ mit $u \neq \pm v \ (mod\ n)$ und es besteht eine gewisse Wahrscheinlichkeit einen nichttrivialen Teiler zu finden.

QS

Das *quadratische Sieb* (QS) ist ein Faktorisierungsalgorithmus, der Lösungen für $u^2 \equiv v^2 (mod\ n)$ findet. Dazu „siebt“ man aus systematisch erzeugten $x_i^2 - n$ solche heraus, die sich zu einer Lösung multiplizieren lassen. Hat man beispielsweise

$$x_1^2 - n = p_1 \cdot p_2 \ , \ x_2^2 - n = p_2^5 \cdot p_3 \ \text{ und } \ x_3^2 - n = p_3 \cdot p_1$$

für drei Primzahlen p_1, p_2, p_3 gefunden, dann folgt aus

$$(x_1^2 - n)\cdot(x_2^2 - n)\cdot(x_3^2 - n) = p_1 p_2 \cdot p_2^5 p_3 \cdot p_3 p_1$$

die gesuchte Lösung $(x_1 x_2 x_3)^2 \equiv (p_1 p_2^3 p_3)^2 \ (mod\ n)$.

Def. *B-glatt*. Sei *B* eine Teilmenge der Primzahlen. Eine natürliche Zahl ist *B-glatt*, wenn alle ihre Primfaktoren in *B* enthalten sind. (Anmerkung: Die gängige Definition für *"x k-glatt"* ist, kein Primfaktor von *x* ist größer als *k*. Die Menge *B* enthält in diesem Fall also alle Primzahlen kleiner oder gleich *k*.)

In unserem Beispiel war also die „Faktorbasis“ $B = \{p_1, p_2, p_3\}$ und x_1, x_2, x_3 sind *B*-glatt. Offensichtlich benötigt das QS eine geeignete Faktorbasis *B* und effiziente Verfahren zum Finden von *B*-glatten $x_i^2 - n$ einerseits (Sieb) und zum Kombinieren solcher zu Quadraten andererseits (Elimination).

Die Faktorbasis

Sei *p* ein Primfaktor von x^2-n , also $p|x^2-n$, dann ist $x^2\equiv n\ (mod\ p)$. Da *x* existiert, muß *n* also quadratischer Rest modulo *p* sein. Die Faktorbasis *B* muß also nur Primzahlen *p* enthalten, für die das Legendre-Symbol $\left(\frac{n}{p}\right)=+1$ ist. Die Wahrscheinlichkeit dafür ist etwa $\frac{1}{2}$, denn es gibt $\frac{p+1}{2}$ quadratische Reste modulo *p*. (Das Legendre-Symbol ist als verallgemeinertes Jacobi-Symbol implementiert).

Es ist einfacher *B*-glatte Zahlen zu finden, wenn die Faktoren der Faktorbasis eher klein sind, daher wählt man $B=\{-1\}\cup\{p\mid p\ prim\wedge p\leqslant p_{max}\wedge\left(\frac{n}{p}\right)=+1\}$.

Die -1 erweist sich als sinnvoll, denn wir erweitern damit den Begriff *B-glatt* auf alle ganze Zahlen. Auch ist immer $2\in B$, denn $0=0^2$ und $1=1^2$. (Anmerkung: Nun kann sich für ein gegebenes n eine Faktorbasis ergeben, in der sehr viele kleinere Primzahlen fehlen. Dadurch sinkt die Wahrscheinlichkeit, dass ein x^2-n *B*-glatt ist. Multipliziert man n nun mit einer kleinen ungeraden Primzahl, ergibt sich möglicherweise eine günstigere Faktorbasis, die den Nachteil, dass *n* nun etwas größer ist, mehr als kompensiert. Siehe dazu density.)

Wie groß soll man die Faktorbasis wählen? Je größer *B* ist, desto leichter ist es *B*-glatte Zahlen zu finden, desto mehr *B*-glatte Zahlen benötigt man aber und umso rechenintensiver wird es daraus ein Quadrat zu multiplizieren. In der Literatur findet man $\#B_{opt}\approx e^{\sqrt{2}/4\cdot\sqrt{\ln n\cdot\ln\ln n}}$, wobei der aktuelle optimale Wert (factor-base-size) implementationsabhängig ist und letztendlich durch den zur Verfügung stehenden Speicher begrenzt ist. Uns interessieren hier nur die Größenordnungen:

Für $n\approx 10^{30},\ 10^{50},\ 10^{70}$ benötigt man etwa $\#B\approx 400,\ 4\ 000,\ 25\ 000$ Faktoren.

Das Sieb

Je kleiner $|x_i^2-n|$ ist, umso wahrscheinlicher ist x_i^2-n *B*-glatt. Also suchen wir in der Nähe von $x\approx\sqrt{n}$. Nun könnte man z.B. für alle *x* in einem geeigneten Intervall $X:=[[\sqrt{n}]-M;[\sqrt{n}]+M]$ durch Probedivision mit den Elementen der Faktorbasis die *B*-glatten x_i^2-n finden. Das ist zu langsam, und der wichtigste Trick des QS ist, vor der Probedivision mit einer schlauen Heuristik solche *x* auszusieben, die mit hoher Wahrscheinlichkeit *B*-glatte x^2-n erzeugen.

Das Sieb *S* wird zunächst für alle $x\in X$ initialisiert: $S_x\leftarrow[\log_2|x^2-n|]$. Das sind kleine ganze Zahlen, und wir realisieren ilog2 durch die LISP-Funktion integer-length. Statt aufwändig zu dividieren subtrahieren wir auf ganze Zahlen abgerundete Logarithmen. Mit einem kleinen zahlentheoretischen Trick lassen sich nämlich für jedes $p_j\in B$ alle $x\in X$ ermitteln, für die $p_j|x^2-n$ gilt.

Angenommen wir kennen eine Zahl r_j mit $p_j \mid r_j^2 - n$. Dann ist für alle $z \in \mathbb{Z}$ auch $(r_j + z \cdot p_j)^2 - n = r_j^2 - n + (2zr_j + z^2 p_j) \cdot p_j$ durch p_j teilbar.

Für alle $x \in X \cap \{r_j + z \cdot p_j | z \in \mathbb{Z}\}$ gilt also $p_j \mid x^2 - n$ und wir setzen an diesen $[(2M+1)/p_j]$ Positionen $S_x \leftarrow S_x - [\log_2 p_j]$. Das läßt sich effektiv durchführen, man beachte, dass lediglich Additionen mit ganzen Zahlen stattfinden. Nachdem man alle $p_j \in B \setminus \{-1\}$ auf diese Art "eingesiebt" hat, ist es einfach, gute Kandidaten für *B*-glatte $x^2 - n$ zu finden, denn dort ist $S_x \approx 0$. In der Praxis durchsucht man das Sieb nach Positionen, an denen S_x kleiner als ein gewisser Schwellenwert (threshold) ist, für den $[\log_2 max\, B]$ eine gute Heuristik ist. Ob diese $x^2 - n$ tatsächlich *B*-glatt sind, ermittelt man dann durch Probedivision.

Nun ist es so, dass kleine Faktoren p_j beim Einsieben die größte Mühe machen, andererseits nur einen geringen Effekt auf S_x haben. Es hat sich herausgestellt, dass es günstig ist, p_j unter einer gewissen Schranke (omit-below), nicht einzusieben, und stattdessen den Schwellenwert etwas größer zu wählen. Außerdem sei hier erwähnt, dass man natürlich auch Potenzen der p_j einsieben könnte, aber der Rechenaufwand steht auch hier in keinem Verhältnis zum Nutzen.

Wie findet man nun ein r_j , das $r_j^2 \equiv n \,(mod\, p_j)$ erfüllt? r_j existiert, denn, wir erinnern uns, *n* ist quadratischer Rest modulo p_j . Statt einfach $0, 1, \ldots, p_j - 1$ auszuprobieren, benutzen wir den genialen *Shanks-Tonelli-Algorithmus*. Er basiert auf einer schrittweisen "Verbesserung" einer angenommen Zahl und wurde als sqrtmododdprime bzw. sqrtmodprime implementiert.

De facto exisitieren für jede ungerade Primzahl p_j zwei *modulare Quadratwurzeln* von *n*, nämlich ein mit dem Shanks-Tonelli-Algorithmus gefundenes r_j und $-r_j \equiv p_j - r_j \,(mod\, p_j)$ und man sollte gehörigst auch beide Lösungen einsieben.

Die Elimination

Wie läßt sich nun aus einer Menge von *B*-glatten Zahlen ein Quadrat zurechtmultiplizieren? Offensichtlich müssen alle Primzahlexponenten gerade werden. Die Gleichungen, die wir ausgesiebt haben, haben die Form $x_i^2 - n = \prod_{p_j \in B} p_j^{e_{i,j}}$. Uns interessiert zunächst nur die Kongruenz $x_i^2 \equiv \prod_{p_j \in B} p_j^{e_{i,j}} (mod\, n)$. Multiplizieren wir nun z.B. die Kongruenzen für x_1 und x_2 , erhalten wir

$$(x_1 x_2)^2 \equiv \prod_{p_j \in B} p_j^{e_{1,j} + e_{2,j}} (mod\, n) \quad .$$

Um herauszufinden, ob ein Primzahlexponent $e_{1,j} + e_{2,j}$ gerade ist, müssen wir lediglich wissen, ob $e_{1,j}$ gerade ist und ob $e_{2,j}$ gerade ist, d.h. uns interessieren eigentlich nur die $e_{i,j} mod\, 2$ und die Multiplikation zweier Kongruenzen wird auf der

rechten Seite zur Addition zweier Vektoren aus $\mathbb{Z}_2^{|B|}$. Das klingt nach Informationsverlust, aber die rechte Seite läßt sich ja aus x_1 und x_2 rekonstruieren (Wir merken uns x_1 und x_2 in xarray). Somit ergibt sich ein lineares Gleichungssystem, das sich mit dem Gaussschen Eliminationsverfahren lösen läßt und allmählich dämmert uns, dass wir etwa $\#B$ Gleichungen benötigen, damit alle Primzahlexponenten gerade werden.

Anders formuliert: Wir wandeln jede ausgesiebte Gleichung $x_i^2 - n = \prod_{p_j \in B} p_j^{e_{i,j}}$ in eine Relation der Form

$$\langle \{i\} \,;\, \{ j \mid e_{i,j} \bmod 2 = 1 \} \rangle$$

um, wir merken uns also nur die "unquadratischen Primfaktoren". Die Multiplikation zweier Gleichungen entspricht dann dieser Verknüpfung zweier Relationen:

$$\langle I_1 ; J_1 \rangle \times \langle I_2 ; J_2 \rangle = \langle I_1 * I_2 \,;\, J_1 * J_2 \rangle \quad ,$$

wobei mit $M_1 * M_2 := (M_1 \cup M_2) \setminus (M_1 \cap M_2)$ die Exklusiv-Oder-Verknüpfung der Mengen M_1 und M_2 bezeichnet wird (isomorph der Addition im $\mathbb{Z}_2^{|B|}$).

Mein einfacher Algorithmus put-into-its-bucket ist aus dem Gaussschen Eliminationsverfahren abgeleitet. Jedem $p_j \in B$ wird ein leerer Eimer $bucket_j \leftarrow nil$ zugeordnet. Jede neue Relation wird in "ihren" Eimer geworfen, entsprechend des größten Primfaktors, der einen ungeraden Exponenten hat. Ist der Eimer bereits besetzt, wird die Relation mit dem Inhalt des Eimers verknüpft. Dadurch verschwindet der ungerade Primzahlexponent und wir wiederholen den Vorgang mit der so gewonnenen Relation.

put-into-its-bucket $(\langle I ; J \rangle)$:
    if $J = \{\}$ then try-to-solve (I)
    else if $bucket_{max\,J} = nil$ then $bucket_{max\,J} \leftarrow \langle I ; J \rangle$
    else put-into-its-bucket $(bucket_{max\,J} \times \langle I ; J \rangle)$

Spätestens wenn alle Eimer voll sind, führt das Hinzufügen einer weiteren Relation dazu, dass alle Primzahlexponenten gerade werden und wir erhalten eine Relation $\langle I ; \{\} \rangle$, die der gesuchten Kongruenz $u^2 \equiv v^2 \,(mod\ n)$ entspricht. Es ist dann

$$u = \prod_{i \in I} x_i \quad \text{und} \quad v = \sqrt{\prod_{i \in I} (x_i^2 - n)} \in \mathbb{N}$$

Man beachte, dass hier eine Quadratwurzel aus einer sehr großen Zahl gezogen wird. Sollte nun $gcd(u \pm v, n)$ nur triviale Teiler von *n* ergeben, erhalten wir mit jeder weiteren Relation eine neue Chance.

In meiner Implementation wird die Suche nach Teilern von *n* erst dann beendet, wenn *n* komplett faktorisiert wurde. Dabei wird auf neu gewonnene Teiler (vgl. new-factor) ein probabilistischer Primzahltest (primeq) angewendet. Zusammengesetzte Teiler lassen sich möglicherweise durch gcd-Verknüpfung mit bereits bekannten

Teilern bzw. durch Untersuchung, ob es sich gar um eine Potenz handelt (power-breaker), stärker faktorisieren.

Die Relationen werden in meiner Implementation nicht durch Bit-Vektoren aus $\mathbb{Z}_2^{|B|}$ repräsentiert. Diese Vektoren sind für neu gewonnene Relationen extrem dünn besetzt und eine Darstellung als Mengen, repräsentiert durch sortierte Listen erschien mir effektiver und ästhetischer. Die Faktoren fallen bei der Probedivision bereits in sortierter Reihenfolge an, und der oben beschriebene Algorithmus wird direkt unterstützt, weil $max\, J$ nicht erst gesucht werden muss, denn es ist ja das erste Element der Liste. Auch läßt sich die Exklusiv-Oder-Verknüpfung zweier sortierter Listen mit destruktiven Listenoperationen sehr effektiv umsetzen (xor-merge). In der Literatur wird erwähnt, dass die Elimination nicht mehr als 30% der Gesamtlaufzeit verbrauchen sollte, dieser Wert wird unterboten.

Partielle Relationen

Wenn wir den Schwellenwert des Siebs zu niedrig wählen, werden uns einige Relationen durch die Lappen gehen. Wählen wir den Schwellenwert dagegen zu hoch, verschwenden wir viel Zeit mit Probedivisionen auf nicht *B*-glatte Zahlen. Bereits in den ersten Implementationen des QS wurde ein Trick verwendet: Bleibt bei der Probedivision von $x_1^2 - n$ nur ein "kleiner" Rest $R \notin B$, also

$$x_1^2 - n = R \cdot \prod_{p_j \in B} p_j^{e_{1,j}} \quad ,$$

so merken wir uns diese "partielle Relation". Finden wir nun einen weitere partielle Relation mit gleichem Rest, dann können wir beide Relationen verknüpfen und *R* unter den Tisch fallen lassen, denn es wird quadratisch:

$$(x_1 x_2)^2 \equiv R^2 \cdot \prod_{p_j \in B} p_j^{e_{1,j}+e_{2,j}} \pmod{n} \quad .$$

Im Prinzip haben wir die Faktorbasis damit um das Element R^2 erweitert. Da es quadratisch ist, spielt es bei der Elimination mit dem Ziel eine quadratische Kongruenz zu finden keine Rolle.

Diese Erweiterung heißt *QS with one large prime*. Tatsächlich ist es aber egal, ob *R* prim ist, andererseits können wir das mit Sicherheit behaupten, wenn $R < (max\, B)^2$, denn *B* hat ja definitionsgemäß keine "Lücken".

In meiner Implementation werden partielle Relationen mit Rest *R* kleiner als slp-max (slp meint *single large prime*) in einer hash-table mit 2^h Plätzen gespeichert. Als hash-Funktion dient ein einfaches $[R/2] mod\, 2^h$. Bei Kollisionen, die keine vollständige Relation ergeben, wird nur die partielle Relation mit dem kleineren Rest *R* behalten, da die Wahrscheinlichkeit für diese einen passenden Partner zu finden, größer ist.

Je größer *n* ist, desto häufiger werden Relationen durch Verknüpfung partieller Relationen gewonnen. Bei sehr großen Zahlen $n > 10^{100}$ wird man praktisch kaum noch vollständige Relationen aussieben, und ist ganz auf diese "partials" angewiesen. Implementationen in dieser Wettkampfklasse gehen deswegen einen konsequenten Schritt weiter und erlauben größere Reste *R*, die ihrerseits faktorisiert werden: Beim *QS with two large primes* werden auch "partial-partials" gespeichert und aus drei solchen Relationen mit Resten $R_1 = q_1 q_2$, $R_2 = q_2 q_3$, $R_3 = q_3 q_1$, die man mit einem graph-cycle-finding Algorithmus findet, läßt sich das gewünschte Quadrat

$$R_1 R_2 R_3 = (q_1 q_2 q_3)^2$$

konstruieren (man greift das Problem sozusagen von zwei Seiten an...). Es sei hier noch erwähnt, dass der Schwellenwert des Siebs optimal eingestellt werden muß, und dass es eine Implementation des *HMPQS with two large primes* gibt, die den cross-over-point zum GNFS vermutlich auf 130 Dezimalstellen nach oben verschoben hat.

Die Größe des Siebs

In Andeutung der allgemeinen Heuristik für Suchalgorithmen, "man suche als erstes, wo man glaubt etwas zu finden", haben wir bereits oben das Suchintervall

$$X := [\,[\sqrt{n}\,] - M\,;\,[\sqrt{n}\,] + M\,]$$

beschrieben. Wie groß muß *M* sein, damit man etwas mehr als $\#B$ Relationen findet? In der Literatur findet man $M \approx \#B^3$. Für die konkrete Implementation ist das unwichtig, man durchsiebt z.B. nacheinander für $k = 0,\ 1,\ 2,\ ...$ kleine Teilintervalle

$$X_k := [\,[\sqrt{n}\,] + (-1)^k \cdot [\,k/2\,] \cdot m\ ;\ [\sqrt{n}\,] + (-1)^k \cdot ([\,k/2\,] + 1) \cdot m - 1\,]$$

der Größe *2m* abwechselnd unterhalb und oberhalb von $[\sqrt{n}\,]$ mit größer werdender Entfernung. Im ungünstigsten Fall hat man dann etwas weniger als ein Teilintervall mehr durchsiebt als nötig ist, um ausreichend Relationen zu ernten, die zu einer Faktorisierung von *n* führen. (Die meisten Implementationen des QS sieben zunächst alle Relationen aus, bevor mit der Elimination begonnen wird. Dann ist man auf eine probabilistische Abschätzung für die Anzahl der benötigten Relationen angewiesen, z.B. $\#B + 20$. Da ich neu gewonnene Relationen sofort verarbeite, entfällt diese Notwendigkeit.

Mit der Zeit entfernt sich nun *x* immer mehr von $[\sqrt{n}\,]$ und $x^2 - n$ wird immer größer und immer seltener *B*-glatt. Gibt es eine Möglichkeit, die $x^2 - n$ irgendwie kleiner zu machen?

MPQS

Das QS sucht nach $x \in I$ mit B-glatten $f(x)$, wobei $f(x)=x^2-n$ .Das Prinzip $p_j | f(s_j) \Rightarrow p_j | f(s_j + z \cdot p_j)$ für alle $z \in \mathbb{Z}$ gilt allerdings auch für jedes andere Polynom mit ganzzahligen Koeffizienten (ungeachtet zunächst der Schwierigkeit, die es möglicherweise bereitet, die passenden s_j zu finden).

Betrachten wir nun das Polynom $g(x)=(ax+b)^2-n$, das Quadrate modulo *n* ganz im Sinne des QS produziert. Wenn wir zu einem gegebenen a nun ein b wählen, so dass $a | b^2-n$, dann ist $g(x)=a^2x^2+2abx+b^2-n$ für alle $x \in \mathbb{Z}$ durch a teilbar. Das Polynom $h(x) := g(x)/a$ wächst zwar stärker als $f(x)=x^2-n$, aber da man durch Wahl anderer a, b praktisch beliebig viele verschiedene Polynome erzeugen kann, kann man das Polynom wechseln statt das Intervall zu vergrößern, um ausreichend Relationen zu sieben. Das ist die Idee des *Multi Polynomial Quadratic Sieve*, kurz *MPQS*.

Betrachten wir nochmal $g(x)$. Sei $a \ll n$ und $b \ll n$. Dann ist $g(x)$ eine ziemlich schlanke Parabel mit einem Minimum etwa bei $g(0)=b^2-n \approx -n$. Damit im gesamten Siebintervall $[-M ; +M]$ etwa gleich große Funktionswerte entstehen, wählen wir nun a so groß, dass wir als Maximum $g(M)=(a \cdot M+b)^2-n \approx +n$ erhalten. Damit ergibt sich

$$a_{ideal} \approx \frac{\sqrt{2n}}{M} \quad .$$

Beim QS durchsieben wir Zahlen der Größenordnung $f([\sqrt{n}]+M) \approx 2 \cdot M \cdot \sqrt{n}$, für das MPQS mit $a \approx a_{ideal}$ dagegen $h(M) \approx n/a \approx \frac{\sqrt{2}}{2} \cdot M \cdot \sqrt{n}$. Während man beim QS mit $M \approx \#B^3$ festgelegt ist, können wir bei gleicher Faktorbasis mit dem MPQS durch Wahl eines kleineren *M*, kleinere Zahlen durchsieben, und entsprechend schneller findet man Relationen.

In der ursprünglichen Version des MPQS ist $a \notin B$ und, damit man es bei den ausgesiebten Relationen nicht berücksichtigen muß, wählt man $a=t^2$ selbst zum Quadrat einer geeigneten Primzahl t . Beim HMPQS, das wir weiter unten eingehend beschreiben, sind die a dagegen *B*-glatt und gehen in die Elimination ein.

Der Nachteil des MPQS ist, dass mit jedem neuen $a \approx a_{ideal}$ die s_j für die gesamte Faktorbasis neu berechnet werden müssen. Nach einem kleinen zahlentheoretischen Exkurs wenden wir uns direkt dem HMPQS zu, das zu einem *B*-glatten *a* erstens eine größere Menge von passenden *b* und damit verschiedene $g_b(x)$ erzeugt, und zweitens innerhalb dieser Menge von Polynomen eine schnelle Berechnung der "Nullstellen" s_j erlaubt.

Modulare Quadratwurzeln

Beim QS benötigten wir Lösungen r_j für $r_j^2 \equiv n (mod\ p_j)$. In diesem Zusammenhang wurde der Shanks-Tonelli-Algorithmus zum Ziehen modularer Quadratwurzeln modulo einer Primzahl erwähnt. Für ungerade Primzahlen gibt es zwei Lösungen, $\pm r_j$.

Beim MPQS müssen wir wegen der Bedingung $a \mid b^2 - n$ zunächst ebenfalls eine quadratische Kongruenz $b^2 \equiv n\ (mod\ a)$ für einen gegebenen Koeffizienten *a* lösen. Wählt man $a = t^2$ als Primzahlquadrat erhält man ebenfalls zwei Lösungen für *b.*

Nehmen wir nun an, $a = \prod a_k$ sei zusammengesetzt mit $a_k\ prim$, $a_k \neq 2$ und $k \neq l \Rightarrow a_k \neq a_l$ Welche *b* erfüllen nun $b^2 \equiv n\ (mod\ a)$?

Aus $b^2 \equiv n\ (mod \prod a_k)$ folgt für alle *k*: $b^2 \equiv n\ (mod\ a_k)$ und wir erkennen, dass für jedes a_k dieselbe Bedingung gilt, wie für die Primfaktoren der Faktorbasis, nämlich $\left(\frac{n}{a_k}\right) = +1$ und dank Shanks-Tonelli können wir die beiden Lösungen $\pm\beta_k$ von $\beta_k^2 \equiv n\ (mod\ a_k)$ für alle *k* bestimmen. Mit Hilfe des chinesischen Restsatzes konstruieren wir nun aus den β_k das gesuchte

$$b \equiv \sum \beta_k \cdot \bar{a}_k \cdot \frac{a}{a_k}\ \ (mod\ a) \quad \text{mit} \quad \bar{a}_k \cdot \frac{a}{a_k} \equiv 1\ \ (mod\ a_k)$$

wobei $\bar{a}_k$ das multiplikative Inverse von $\frac{a}{a_k}$ modulo a_k ist, welches existiert, da a_k und $\frac{a}{a_k}$ teilerfremd sind (zur Berechnung verwendet man einen modifizierten Euklidischen Algorithmus, siehe invmod).

Nehmen wir nun an, *a* hätte $d+1$ Primfaktoren, also $k \in \{0, 1, \ldots, d\}$, dann ergeben sich wegen der Vielfachheit der β_k insgesamt 2^{d+1} verschiedene

$$b(\mu) \equiv \sum_{i=0}^{d} \mu_k \cdot b_k\ \ (mod\ a) \quad \text{mit} \quad b_k := \beta_k \cdot \bar{a}_k \cdot \frac{a}{a_k} \quad \text{und} \quad \mu \in \{-1, +1\}^{d+1}$$

HMPQS

Diesen Zusammenhang nutzt das HMPQS aus, denn entsprechend viele unterschiedliche $g_b(x)$ können wir für ein geeigenetes *a* in einem Abwasch konstruieren. Betrachten wir nun μ und ν , wobei $\nu_k = -\mu_k$ für alle $k \in \{0, 1, \ldots, d\}$ sein soll. Offensichtlich ist $b(\mu) = -b(\nu)$. Leider ist

$$g_{-b}(x) = (a \cdot x - b)^2 - n = (a \cdot (-x) + b)^2 - n = g_b(-x)$$

und da wir in einem Intervall $x \in [-M\ ;+M]$ sieben, würden wir alles doppelt durchsieben. Um das zu vermeiden, müssen wir verhindern, dass alle Vorzeichen μ_k alternieren und wir setzen deswegen $\mu_0 := +1$ und es verbleiben uns immerhin 2^d verschiedene Polynome $g_b(x)$.

Um möglichst viele Polynome zu erhalten, sollte *d* also möglichst groß sein, *a* hat also möglichst viele kleine Primfaktoren a_k , und wegen $\left(\frac{n}{a_k}\right)=+1$ ist *a* *B*-glatt und jedes a_k teilt $g_b(x)$ für alle $x\in\mathbb{Z}$.

Es bleiben zwei Fragen offen. Erstens: Wie berechnet man die s_j für ein $g_b(x)$? Zweitens: Wie wählt man *d* und die $a_k\in B$, so dass $a=\prod_{k=0}^{d} a_k\approx a_{ideal}$?

Der Hyperwürfel

Im Abschnitt "das Sieb" hatten wir r_j mit $r_j^2\equiv n\,(mod\ p_j)$ definiert. Beim QS mußte man sie für alle $p_j\in B$ berechnen, wegen ihrer nützlichen Eigenschaft $p_j\mid f(\pm r_j+z\cdot p_j)$ für $z\in\mathbb{Z}$. Beim HMPQS benötigen wir sie zunächst als β_k , denn $\beta_k=r_j$ für $a_k=p_j\in B$. Außerdem brauchen wir sie zur Bestimmung der s_j mit der nützlichen Eigenschaft $p_j\mid g(s_j+z\cdot p_j)$. Aus $g(x)=f(ax+b)$ folgt nämlich direkt $a\cdot s_j+b\equiv r_j\,(mod\ p_j)$. Um diese Kongruenz nach s_j aufzulösen, benötigen wir das multiplikative Inverse von *a* modulo p_j , nennen wir es $\mathring{a}_j$ mit $a\cdot\mathring{a}_j\equiv 1\,(mod\ p_j)$. Dieses Inverse existiert aber nur, wenn *a* und p_j teilerfremd sind, was nicht weiter stört, denn wenn $p_j=a_k$ ist, wissen wir ja ohnehin, dass $p_j\mid g(x)$, denn so hatten wir $g(x)$ ja konstruiert (In meiner Implementation setze ich `factor-1/amodp` $\leftarrow -1$, als Markierung für die $p_j=a_k$, die nicht eingesiebt werden). Wir erhalten schließlich zwei Lösungen

$$s_j\equiv\mathring{a}_j\cdot(\pm r_j-b)\,(mod\ p_j)\ .$$

Für jedes μ müssen wir zunächst $b(\mu)$ und dann für die gesamte Faktorbasis die s_j berechnen, bevor wir sieben können. Ein nicht zu unterschätzender Rechenaufwand! Bereits die beiden unabhängigen Erfinderteams des HMPQS haben sich hier Gedanken gemacht: Jedes μ mit $\mu_0=+1$ entspricht der Ecke eines *d*-dimensionalen Hyperwürfels der Kantenlänge 2 und die umständliche Summenberechnung für $b(\mu)$ wird zu einer einfachen Addition, wenn wir nur systematisch von einer Ecke zur nächsten wandern, also jeweils das Vorzeichen für nur ein μ_k wechseln. Wir erreichen jede Ecke des Hyperwürfels, wenn *k* einem Hamiltonschen Pfad folgt:

path(d): if d=0 then [ ] else path(d-1) & [d] & path(d-1)

z.B. path(4)=[1 2 1 3 1 2 1 4 1 2 1 3 1 2 1]

Sei also $\mu'_k=-\mu_k$ und $\mu'_{i\neq k}=\mu_i$, dann ist $b(\mu')\equiv b(\mu)-2\cdot\mu_k\cdot b_k\ (mod\ a)$. Insbesondere ergibt sich $s_j(\mu')\equiv s_j(\mu)+\mu_k\cdot 2\mathring{a}_j b_k\,(mod\ p_j)$ und es lohnt sich die Werte $2\mathring{a}_j b_k$ in einem zweidimensionalen array (`kip`) zu speichern, bevor man den Hyperwürfel traversiert.

Der *a*-Generator

Wie konstruiert man nun die *a* mit $a=\prod_{k=0}^{d} a_k \approx a_{ideal}$ und $a_k \in B$? *d* soll möglichst groß sein, die ausgewählten a_k also möglichst klein. Wir erinnern uns: Die $p_j = a_k$ werden nicht eingesiebt. Wenn wir jetzt die kleinsten $p_j \in B$ als a_k verwenden, erzeugt das Polynom $h(x)$ vermutlich seltener *B*-glatte Zahlen (siehe die Anmerkung im Abschnitt zur Faktorbasis). *d* sollte aber auch nur so groß sein, daß man wenigstens einige *a* benötigt, um *n* zu faktorisieren, damit sich der Aufwand den Hyperwürfel für *a* vorzubereiten gelohnt hat (siehe analog dazu das Argument der Teilintervalle im Abschnitt zur Größe des Siebs).

Ich verwende daher nur a_k mit $a_k = p_j$ und $2q_{min} \leqslant j < 2q_{max}$. Innerhalb dieser Schranken gibt es also $\bar{q} = q_{max} - q_{min}$ gerade und ebensoviele ungerade *j*. Mit je drei dieser ungeraden *j*, erzeuge ich nun alle $\binom{\bar{q}}{3}$ möglichen Produkte

$$top := p_{j_1} \cdot p_{j_2} \cdot p_{j_3} \quad \text{mit} \quad j_1 \neq j_2 \neq j_3 \neq j_1 \quad .$$

Das kleinste der Produkte wäre somit $top_{min} = p_{2q_{min}+1} \cdot p_{2q_{min}+3} \cdot p_{2q_{min}+5}$. Diese Tripel verteile ich nach ihrem Logarithmus auf einen array *A* der Größe *T*, wobei jedes Element mehrere Tripel enthalten kann:

$$A[index(\log top)] \leftarrow top \quad \text{mit} \quad index(x) := [\frac{x - \log top_{min}}{\log top_{max} - \log top_{min}} \cdot T] \quad .$$

Nun erzeuge ich mit einer rekursiven Funktion systematisch aus den geraden *j* alle möglichen Produkte *a*, bis *a* so groß geworden ist, daß es sich mit Elementen von *A* ergänzen läßt, so dass $a \cdot top \approx a_{ideal}$. Dazu startet man make-cubes $(1, q_{min})$.

make-cubes (a, q) :
    if $\log a_{ideal} - \log a > \log top_{min}$ then
        unless $\log a_{ideal} - \log a > \log top_{max}$
            use-cube $(a \cdot A[index(\log a_{ideal} - \log_a)])$
    else if $q < q_{max}$ then
        make-cubes $(a \cdot p_{2q}, q+1)$
        make-cubes $(a, q+1)$

In dieser schematischen Beschreibung wurde vernachlässigt, dass ein Element des arrays leer ist oder mehrere *top* enthält, also kein oder gleich mehrere $a \approx a_{ideal}$ mit use-cube erzeugt werden. Vielmehr sollte hier gezeigt werden, dass make-cube ganz nebenbei die Dimension *d* des Hyperwürfels erzeugt, und dass zuerst Hyperwürfel mit großem *d* erzeugt werden, und dass alle erzeugten *a* verschieden sind, da die beiden Teilmengen von *B* (gerade und ungerade *j*) disjunkt sind.

Allerdings hat es einiges an Experimentieren erfordert, bis ich vernünftige Parameter gefunden habe, so dass der array einerseits nicht zu dünn besetzt ist (damit ausreichend viele *a* enstehen), andererseits aber groß genug ist, so dass der Fehler

$$error(a) := |a - a_{ideal}| / a_{ideal}$$

klein ist. Für $n \geqslant 10^{42}$ verwende ich $q_{min} = 5$, $q_{max} = 105 \ll \frac{\#B}{2}$ und verteile die 161 700 möglichen *top* auf $T = 20\ 000$ und erreiche $error(a) < 0{,}05\,\%$.

Für $n < 10^{42}$ entstehen auf diese Art zu wenig Hyperwürfel, weshalb die *top* dann aus nur zwei Faktoren zusammengesetzt werden. Unterhalb von $n \approx 10^{25}$ ist q_{max} zusätzlich durch $\#B/2$ beschränkt. Dies alles führt zu einer geringeren Anzahl von *tops*, dem mit $T := min\{20\ 000, [\#\{top\}/8]\}$ begegnet wird. Trotzdem erreicht man noch $error(a) \approx 0{,}5\,\%$ und in einem Test mit 10^5 verschiedenen $n \approx 10^{23}$ wurde immer eine für die Faktorisierung ausreichende Menge von *a* erzeugt. Für $n < 10^{23}$ weiche ich auf den Algorithmus make-cublets aus, der auch die kleineren Elemente von *B* hinzuzieht und *error(a)* vernachlässigt. Hier geht es nur noch darum die Lücke zu primitiven Faktorisierungsalgorithmen zu schließen (Probedivision mit den ersten 100 000 Primzahlen genügt bis $n \approx 10^{12}$).

Da man über die Zusammensetzung der Faktorbasis nur probabilistische Aussagen machen kann (die Wahrscheinlichkeit für $\left(\frac{n}{p}\right) = +1$ ist etwa $\frac{1}{2}$), besteht ein prinzipielles Restrisiko, dass make-cubes versagt. Dann wird die Faktorisierung mit der nächstsicheren der drei vorgestellten Methoden fortgesetzt.

Das Sieb im Detail

Um die Indizierung des Siebs effektiver zu gestalten, bilde ich zunächst das Intervall $X := [-M\,;M]$ auf $Y := [0\,;2\mathrm{M}]$ ab. Das Siebpolynom lautet dann

$$h'(y) = h(y - M) = (a\,y + \beta) \cdot y + \gamma$$

$$\text{mit } \beta := 2(b - Ma) \text{ und } \gamma := aM^2 - 2\mathrm{bM} + \frac{b^2 - n}{a} \ ,$$

so dass je zwei Multiplikationen und Additionen mit bignums erforderlich sind, um ein $h'(y)$ zu berechnen. Die Initialisierung des Siebs $S_y \leftarrow [\log_2 |h'(y)|]$ wäre viel zu langsam, wenn wir $h'(y)$ für jedes $y \in [0\,;2\mathrm{M}]$ berechnen würden. Stattdessen bestimme ich für alle in Frage kommenden $l \in \mathbb{N}$ die Teilintervalle $Y_l \subset Y$, so dass $[\log_2 |h'(y)|] = l$ für alle $y \in Y_l$. Allerdings gibt es in jedem der vier Quadranten der Parabel (je nach den Vorzeichen von $h'(y)$ und $y - \frac{\beta}{2\mathrm{a}}$) je ein solches Teilintervall. Außerdem benötigt man die Umkehrfunktion von $h'(y)$, um die Grenzen $h'^{-1}(2^l)$ der Teilintervalle zu bestimmen. Ich benutze Fließkommaarithmetik, um die dafür nötigen Wurzeln zu ziehen.

Nun kann man endlich die p_j mit $p_j \nmid a$ einsieben. Selbstverständlich sollte man beide "Nullstellen" s_j verwenden. Aus $p_j | g(s_j)$ folgt $p_j | h'(s_j + M)$ und bei dem kleinsten $y \in Y$ mit $p_j | h'(y)$ beginnt eine hoffentlich gut geölte Schleife:

$$y \leftarrow s_j + M + \left[\frac{-M - s_j}{p_j}\right] \cdot p_j$$

repeat $\quad S_y \leftarrow S_y - [\log_2 p_j]$

$\quad y \leftarrow y + p_j$

until $\quad y > 2\mathrm{M}$

Beim Aussieben, also $S_y <$ threshold, ist zu beachten, dass wir alle p_j , also auch die mit $p_j | a$, probedividieren müssen, denn $g(y - M) = a \cdot h'(y)$ kann durchaus durch Potenzen eines a_k teilbar sein, nicht zuletzt da die a_k möglichst klein gewählt wurden. Ergibt die Probedivision nun, dass ein $h'(y_i)$ B-glatt ist, also

$$h'(y_i) = h(y_i - M) = \prod p_j^{e_{i,j}}$$

dann haben wir eine Relation gefunden, denn

$$g(y_i - M) = a \cdot h(y_i - M) = (a(y_i - M) + b)^2 - n$$

ist ebenfalls B-glatt. Statt mit a zu multiplizieren, um die a_k anschließend wieder herauszudividieren, schreiben wir die neue Relation lieber gleich so:

$$\langle \{i\} ; \{ j \mid e_{i,j} \bmod 2 = 1 \} * \{ j \mid (\exists k)\ (a_k = p_j) \} \rangle \quad \text{mit} \quad x_i := a(y_i - M) + b$$

und verwenden sie im Übrigen wie im Abschnitt Elimination beschrieben.