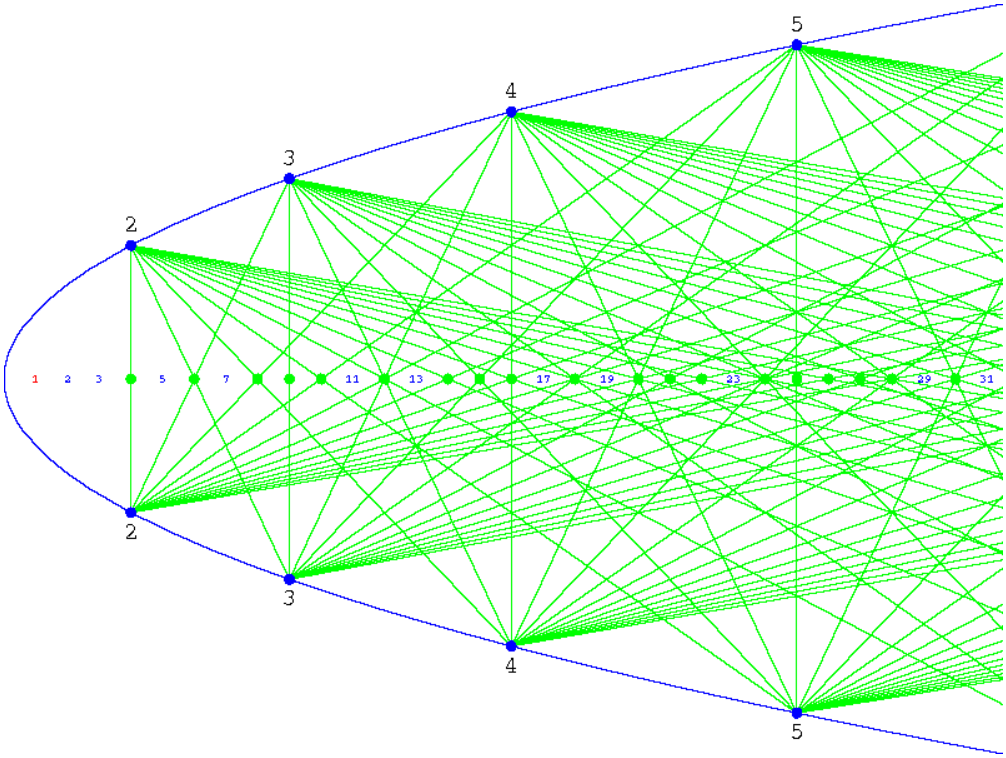
Hans Block, SCB

Faktorisering och samhällets säkerhet

Sammanfattning

Kryptering är absolut nödvändig för säker databehandling. Under 1970-talet utvecklades asymmetriska krypteringsmetoder för säker nyckeldistribution och signering. Den mest använda metoden bygger på att det är svårt dela upp stora tal i primfaktorer. En rad metoder för detta har utvecklats och implementerats. Föredraget beskriver den livliga och intressanta forskningen kring dessa frågor. Slutligen berörs mina egna erfarenheter.



Figur. Matematik är svårt att illustrera. Bilden visar hur primtal blir kvar efter sållning. Den kallas *A visual Sieve for Prime Numbers, invented by Yuri MATIYASEVICH and Boris STECHKIN*, och är hämtad från:
http://logic.pdmi.ras.ru/~yumat/Journal/Sieve/Sieve.html

Innehåll

1 Inledning

The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.

Karl Friedrich Gauss
Disquisiones Arthmeticæ, Art. 329 (1801)

Själv uttrycker jag saken så här: Primtal känner alla till. Resultaten är konkreta. Området är fyllt av upptäckarglädje och skaparlust. Metoderna hämtas från många håll. Utvecklingen är snabb. *Primtal är kul!*

Det finns seriösare skäl att ägna sig åt primtal: Vi är dagligen beroende av teknik. Det hör till allmänbildningen att förstå åtminstone något av den teknik vi omges av. Vi skaffar den kunskapen på olika sätt. Pojkar mekar med mopeder. De lär sig hur motorer ser ut inuti och hur de fungerar. De får en aning om teknikens villkor.

Vi vuxna använder internetbanker och smarta kort. Rutinmässigt, utan att märka något, lottar vi fram primtal och multiplicerar ihop dem. Våra pengars säkerhet bygger på att sammansatta tal är svåra att faktorisera. Att begripa öppna och hemliga nyckelsystem hör till databranschens allmänbildning. Ingen kan undgå primtal. Låt oss leka med dem en stund!

Föredraget skall ge en matematiskt intresserad person inblick i primtalsforskning. Faktorisering bygger på traditioner från tre helt olika områden, som tidigare spelat en ganska undanskymd roll: kryptologi, talteori och långa numeriska beräkningar.

1.1 Kryptologi

Att dölja meddelanden har alltid behövts. Ett tidskrävande sätt var att raka huvudet på en slav, skriva ett meddelande på hans kala hjässa, låta håret växa ut och skicka iväg honom. Caesars chiffer var snabbare: I detta ersattes varje bokstav med den som kommer tre platser efteråt i alfabetet. Systemet dög i en tid då de flesta var analfabeter.

Eftersom systemen var svaga, så var nästan allt hemligt: metoder för kryptering, metoder för forcering och uppnådda resultat.

1.2 Talteori

Talteorin har en lång och lysande historia, från grekisk talmystik, Euklides, Fermat, Gauss, Abel … Det var matematik för dess egen skull, en skön konst, utan sidoblick på fysiska eller ingenjörsmässiga tillämpningar. Talteorin var den garanterat mest onyttiga delen av matematiken!

1.3 Långa numeriska beräkningar

Inom många områden behövdes ”sifferslavar” för svåra numeriska beräkningar. Kalkylering av planetbanor var ett tidigt exempel. Man konstruerade upp till 11-ställiga logaritmtabeller, men dessa var svåra att kontrollera och behäftade med fel.

Primtalsforskarna hade liknande villkor. Arbetet byggde på stora primtals- och faktoriseringstabeller, som framställts under stor möda, inte utan misstag. Trots att den store Gauss hängav sig åt beräkningar såg teoretikerna ofta ner på tabellframställning och numeriskt räknande.

Faktorisering var en konst mer än en teknik. Tur spelade en stor roll.

1.4 Orsaker till förändring

Konstruktion av datorer krävde grundliga studier av det som annars är självklart: hur man utför de fyra räknesätten. Sortering var ett centralt problem som krävde nya metoder. Komplexitetsteorin blev ett naturligt sätt att tänka.

Nya primtalsrekord möjliggjordes först av räknemaskiner och sedan av datorer. Nya metoder blev praktiskt intressanta. ”Faktoriseringskufarna” fick använda gratis beräkningstid på den tidens stora maskiner.

Militärerna insåg, att krypteringsmetoder förr eller senare blir kända av motståndarna – allra senast när de används ute på förbanden. Ett system som avslöjas måste ersättas direkt. Detta gick inte på den civila marknaden, där behoven växte explosionsartat. I stället gick man en annan väg. Gränssnitt mellan applikationer och kryptering standardiserades. Metoderna skulle vara enkla att förstå men komplicerade att forcera. Hela världen inbjöds att knäcka förslaget till Data Encryption Standard. Hemligheten skulle i stället ligga i långa nycklar. Dessa måste kunna distribueras säkert.

Diffie – Hellmans artikel 1976 om asymmetriska metoder möjliggjorde detta. Strax efter kom RSA-metoden, som tidigare funnits några år i den hemliga världen.

Faktorisering blev värd att studera. Forskningen lånade friskt från andra delar av matematiken. Senare har även andra asymmetriska krypteringsmetoder kommit fram.

När datorerna blev större och snabbare hjälpte detta i första hand forcörerna. Krypton knäcktes, och asymptotiskt snabba metoder kunde realiseras och konkurrera ut tidigare kända metoder.

På lång sikt blir det annorlunda. Forcering är alltid komplexare än kryptering och dekryptering. Om båda applikationerna gynnas av samma datorutveckling, så kommer på sikt forceringen bli så mycket tyngre, att alla krypton blir omöjliga att angripa med klassisk forcering.

Primtalsforskning – påpekar Crandall och Pomerance – är ett område, där ingen människa kan känna till allt. Å andra sidan kan man göra väldigt mycket med begränsade kunskaper och barnslig tro på andras resultat. Jag har haft stort nöje av att under lång tid följa utvecklingen.

2 Asymmetriska krypteringsmetoder

2.1 Begrepp

Under senare delen av 1970-talet framkom tekniker som skulle lösa problemen med: *sekretess* (bara sändare och mottagare kan se vad som sänds), *oförvanskbarhet* (mottagaren kan vara säker på att ingen har manipulerat meddelandet) *autenticitet* (meddelandet kommer från den som säger sig ha sänt det) och *oförnekbarhet* (avsändaren kan inte i efterhand säga att han/hon inte har sänt det).

För att uppnå dessa mål behövdes en grundläggande ny teknik, en asymmetrisk kryptering baserad på *en öppen* och *en hemlig nyckel*, som styr funktioner som är varandras inverser, men där man inte kan räkna ut den andra nyckeln, om man har den ena.

Vidare behövs *protokoll* för hur dialogen mellan användarna skall gå till. Av effektivitetsskäl behövs en *hash-algoritm*, som snabbt ger en checksumma på hela meddelandet. Summan skall ha kryptologiska egenskaper, så att ingen kan ändra någon enda bit i meddelandet, utan att checksumman förändras på ett oförutsebart sätt.

2.2 Protokoll

Mycket förenklat löses de fyra grunduppgifterna för ett meddelande *M* som följer. I exemplen är det som alltid i detta ämne Alice och Bob som kommunicerar med varandra.

2.2.1 Sekretess

Alice skall skicka ett meddelande till Bob. Det är långt och asymmetriska metoder är resurskrävande, så Alice lottar fram en sessionsnyckel. Hon krypterar denna med Bobs öppna nyckel. Bob dekrypterar sessionsnyckeln med sin hemliga nyckel. Meddelandet i sin helhet krypteras sedan av en snabb symmetrisk krypteringsalgoritm med hjälp av sessionsnyckeln.

Men hur kan Alice vara säker på att trafiken som ser ut att komma från Bob verkligen kommer från Bob? För att få reda på detta, måste Alice få Bobs öppna nyckel, direkt och krypterad, från någon instans som alla måste lita på. Denna brukar kallas *CA*, Certification Agency.

2.2.2 Oförvanskbarhet

Ett enkelt sätt är att använda en hash-algoritm. Alice som skickar meddelandet räknar ut checksumman och krypterar den med sin hemliga nyckel. Bob läser meddelandet, räknar ut checksumman och jämför den med den dekrypterade checksumman. Om dessa är lika, så är meddelandet oförvanskat.

Hashalgoritmen bör också ha en nyckel som parterna kommer överens om.

2.2.3 Autenticitet

Alice skall tala om för Bob vem hon är. Hon skickar ett certifikat, en krypterad text av vilken det framgår när certifikatet skapats, vem hon är, vem som utfärdat certifikatet och hur länge det gäller. Hon har krypterat det med sin hemliga nyckel. Bob tar kontakt med CA och får reda på Alice's öppna nyckel och om den fortfarande är giltig. Även detta bekräftas genom ett certifikat från CA.

2.2.4 Oförnekbarhet

När Alice har skickat ett meddelande till Bob, så har hon krypterat åtminstone checksumman med sin hemliga nyckel. Om hon skött sin nyckel och inte anmält nyckeln som försvunnen, så kan ingen annan än hon ha skickat meddelandet.

2.3 Förtroendekedjan

Den korta sammanfattningen av protokollen väcker ytterligare frågor:

- Hur vet Alice att en öppen nyckel verkligen tillhör Bob?
- Hur vet hon att den inte har blivit indragen?
- Kan en angripare avlyssna trafiken, snappa upp ett meddelande, och skicka något felaktigt i dess ställe?
- Har Microsoft lagt in bakdörrar? Företaget ser till att kunderna skickar hem meddelanden över Internet om andra uppgifter: maskinkonfiguration, vilka filer som körts av en mediaspelare och annat. Skulle företaget i något läge också kunna skicka ut krypteringsnycklar eller lösenord?
- Är programinstallationen korrekt, eller har viktiga delar av koden kringgåtts?
- Vem kan man lita på inom det egna företaget?
- Finns det någon auktoritet i det svenska samhället eller världssamfundet som alla kan lita på?
- Hur gör staten och bankerna?
- Hur gör internationella organisationer?

Förtroendekedjan har alltså många länkar, av vilka flera kan vara svaga. Säkerhetsmedvetna organisationer har därför mycket att bekymra sig för. En listig motståndare kan förvisso nå sitt mål på många andra sätt är genom att faktorisera stora tal.

Men åt denna speciella risk ägnas denna lilla uppsats: hur man angriper RSA-metoden rent matematiskt.

3 Fermats lilla sats och RSA-metoden

Det finns ett antal asymmetriska krypteringsmetoder. Den i särklass vanligaste är RSA-metoden. Man förstår den med ett minimum av kunskaper.

3.1 Fermats lilla sats

Om p är ett primtal, så är

$$a^{p-1} \equiv 1 (\mathrm{mod}\, p)$$

Satsen kan användas till mycket, bl.a. för att bevisa att tal *inte* är primtal.

Bevis 1.
Restklasserna till talen i intervallet [1 .. p – 1] är en grupp under multiplikation mod p.

Bevis 2.
Genom induktion över a bevisas att

$$a^{p} \equiv a (\mathrm{mod}\, p)$$

3.2 RSA-metoden

Ett meddelande M skrives som ett tal (mod pq), där p och q är primtal. M blir krypterat

$$M^{k} (\mathrm{mod}\, pq)$$

där k är valt så att

$\gcd(k, p-1) = 1$ och $\gcd(k, q-1) = 1$

Dekryptering sker också genom exponentiering modulo pq. Vi väljer dekrypteringsnyckeln d så att

$$\left(M^{k}\right)^{d} (\mathrm{mod}\, pq) = M^{k \cdot d} (\mathrm{mod}\, pq) \equiv M (\mathrm{mod}\, pq)$$

Men enligt kinesiska restsatsen är, om $\gcd(M, pq) = 1$,

$$M^{x} \equiv 1 (\mathrm{mod}\, pq)$$
$$\Leftrightarrow$$
$$\left(M^{x} \equiv 1 (\mathrm{mod}\, p)\right) \wedge \left(M^{x} \equiv 1 (\mathrm{mod}\, q)\right)$$

vilket är sant om $p - 1 \mid x$ och $q - 1 \mid x$ d.v.s.

$$scd((p-1),(q-1)) \mid x$$

(scd: smallest common dividend). Vi skall alltså bestämma *d* så att

$$k \cdot d \equiv 1 \bmod scd((p-1),q-1))$$

och det går, eftersom

gcd(k, p-1,q-1)=1

Vi får alltså tillbaka klartexten. Detta stämmer också då gcd(*M, pq*) > 1.

Om man har *p, q* och *d*, så kan man finna *k*, och om man har *p, q* och *k*, så kan man finna *d*, allt med Euklides' algoritm. Utan att faktorisera *n* kan man inte finna *d* om man har *k* eller *k* om man har *d*.

Omvänt, om *n, k* och *d* är givna, så kan man också i allmänhet snabbt finna *p* och *q*. Att knäcka RSA-kryptering är alltså ungefär ekvivalent med faktorisering.

För att utvärdera metoden måste vi svara på ett antal frågor som rör nyckelutgivare, användare av kryptot och forcörer:

- Hur hittar nyckelutgivaren stora primtal?
- Hur bevisar nyckelutgivaren att ett tal är primtal?
- Hur fort räknar nyckelutgivaren ut största gemensamma delare?
- Hur fort kan användaren exponentiera modulo heltal?
- Hur fort kan angriparen faktorisera stora tal?
- Hur mycket utrymme tar operationerna ovan?

Dessa frågor inbjuder till en lång utflykt inom tal- och komplexitetsteori. Storleken på de tal som hanteras gör att metoder som tidigare endast hade akademiskt intresse nu har implementerats i standardmässiga program.

4 Komplexitet

4.1 Ny intresseinriktning

I matematiken har intressena skiftat. I klassisk matematik värderades *existenssatser*. Ett exempel är algebrans fundamentalsats: Varje polynomekvation har en rot bland de komplexa talen. Inom algebran finns satser som talar om *strukturen* för t.ex. alla ändliga grupper eller kroppar. *Hur* operationerna utfördes var av mindre intresse. Man kunde vara mer explicit och *konstruera* någonting *i ett ändligt antal steg*, t.ex. de irreducibla faktorerna till ett polynom med heltalskoefficienter.

I numerisk analys sökte man metoder för att *beräkna någonting med viss noggrannhet*. Ett exempel: Finn numeriskt rötterna till en polynomekvation. Tiden för konvergens blev intressant. I beräkningsinriktad talteori är nog-

grannheten sällan ett problem, men man är intresserad av algoritmer och *tid* för exekvering och *minnesutrymme*, när någonting går mot oändligheten.

Sedan 60-talet har algoritmteorin utvecklats mycket. Mängden resultat kan verka förvirrande, men antalet tekniker är ändå begränsat.

En gemensam metod är *divide and conquer*, dela och besegra. Ofta kan den få algoritmer att gå från n^2 till $n \cdot \log n$ tid i så skilda fall som sortering, diskreta fouriertransformer och polynommultiplikationer. Detta är alltså standard. Newton – Raphsons gamla approximationsmetod har också fått nya tillämpningar.

4.2 Vad som mäts

4.2.1 Beräkningstid

Tiden bör i första hand förutsägas och i andra hand mätas i en viss implementation.

Mätningen kan ske på många sätt. Det brukar vara lätt att uppskatta antalet *aritmetiska operationer* eller *operationer i någon grupp*. När talen får plats i datorns helord är det lätt att räkna om detta till *maskininstruktioner* eller klockcykler.

När ett tal kräver flera ord för sin lagring, måste man ta hänsyn till hur aritmetiken utförs. Då är det mer meningsfullt att beräkna *bitinstruktioner*, så att man får ett någorlunda maskinoberoende mått.

Den asymptotiska komplexiteten säger inte allt. Mycket kan döljas i symbolen *O(n)*. Uppskattningarna kan förfinas, och man kan även teoretiskt få en god uppfattning om en förbättring på bara en faktor 2.

Ledaren för ett faktoriseringsprojekt är mycket intresserad av kalendertiden för de totala beräkningarna. När man är färdig med programmeringen och skall finslipa algoritmen, så noterar man exekverade *varv i inre loopar*, med syftet att minimera antalet operationer i den mest utförda loopen.

Alla sätten att mäta komplexiteten kan komma ifråga. Till slut är det ändå resultatet som räknas. När en beräkning tar ett halvår, så betalar det sig att åstadkomma även en 10 %-ig förbättring, även om den inte framgår av vackra formler.

4.2.2 Utrymme

Behov av primär- och sekundärminnesutrymme är lika viktigt. Givetvis är det skönt om man kan använda enbart primärminne. Det finns ofta kompromisser mellan förbättringar i tid och utrymme.

Om program och data under det intensiva skedet i sin helhet kan rymmas i processorns casheminne, så kan det öka hastigheten, utan att det framgår av några formler. Några av faktoriseringsalgoritmerna tar extremt litet utrymme och borde kunna optimeras för detta.

4.2.3 Parallellitet

Faktoriseringsalgoritmerna konsumerar extrema maskinresurser. En viktig fråga är hur arbetet kan fördelas mellan många maskiner.

För somliga algoritmer är detta knappas möjligt: Beräkningarna går i en enda följd. I andra fall är det möjligt, men de parallella processorerna kräver gemensamt minne eller i varje fall mycket snabb kommunikation. Detta brukar uttryckas så, att problemet har en fin *granualitet*.

I ytterligare några fall kan processorerna arbeta helt parallellt utan att någon kommunikation dem emellan behövs.

4.2.4 Kostnad

Ovanstående faktorer bör vägas samman till en kostnad. Daniel J. Bernstein beräknar kostnaden för en faktorisering som produkten av beräkningstiden och kostnaden för de maskiner på vilka algoritmen exekveras. Han utgår också från att många uppgifter kan lösas lika enkelt genom att designa en krets som att skriva ett program. Han anser också, att kostnaden för en maskin beror mera på *antalet* komponenter än på deras *typ* – minnen, processorer eller specialdesignade kretsar – i varje fall när antalet delar går mot oändligheten.

Detta modell gynnar algoritmer med stora beräkningar på litet utrymme.

4.3 Några fundamentala algoritmer

Vi skall nu undersöka några komponenter i faktoriseringsalgoritmerna.

4.3.1 Multiplikation

På lågstadiet får vi lära oss att skriva våra räkningar så här:

```
      xxxx
      yyyy
  .________
     zzzzz
    zzzzz
   zzzzz
  zzzzz
 zzzzz
_________
  uuuuuuuuu
```

Romben har ytan och algoritmen har tidskomplexiteten $\log^2 n$.

En enkel förbättring kan göras genom att vi klyver operanderna i två delar om vardera j bitar:

$$u = 2^j \cdot U_1 + U_0,\ v = 2^j \cdot V_1 + V_0$$

$$uv = \left(2^{2j} + 2^j\right)\cdot U_1 \cdot V_1 + 2^j \cdot \left(U_1 - U_0\right)\cdot\left(V_0 - V_1\right) + \left(2^j + 1\right)\cdot U_0 \cdot V_0$$

Därmed får vi tre multiplikationer i stället för fyra! Detta kan tillämpas rekursivt. Om *n* har 2^k delar, så blir det $s = 3^k$ multiplikationer.

$$\lg s = \lg 3 \cdot k = \lg 3 \cdot \lg\lg n$$

$$s = 2^{\lg\lg n \cdot \lg 3} = \left(\lg n\right)^{\lg 3} \approx \left(\lg n\right)^{1{,}585}$$

Komplexiteten har alltså tagit ett blygsamt steg neråt.

Ett annat sätt att multiplicera snabbt är modulär aritmetik.

Ett bättre medel är *fouriertransformer*. Några korta ord om dem: Multiplikation av två stora tal innebär en faltning av siffrorna. Fouriertransformer omvandlar som bekant faltning till elementvis multiplikation och omvänt. Den ändliga fouriertransformen tar utan eftertanke m^2 flytande multiplikationer, där $m = \lceil \lg n \rceil / b$ är antalet ord som talet lagras i och *b* är antal bitar/ord. I den snabba fouriertransformen samlar man udda och jämna termer för sig och får två problem som är hälften så stora som det ursprungliga. Detta görs rekursivt. Resultatet blir $O(m \cdot \lg m)$ operationer. Detta är avgörande, eftersom det bara behövs *m* elementvisa multiplikationer.

Det behövs ganska stora tal, kanske fler än 60 siffror, för att avancerade metoder skall löna sig. Men redan RSA-metoden använder nycklar om 2000 bitar. Snabb multiplikation är ännu mera nödvändig om man skall hitta rekordstora primtal. Paket för avancerad aritmetik finns på nätet.

4.3.2 Division, mod

Division kan använda snabb multiplikation. Genom att räkna ut 1/*n* i förväg kan man få fram flera *a* mod *n* med samma *n*.

4.3.3 Exponentiering mod *n*

Exponentiering löses med successiva kvadreringar. Bitarna i exponenten avläses framlänges eller baklänges. En exponent *k* löses på maximalt 2·lg *k* multiplikationer.

Tid för RSA-kryptering blir därför högst $O(\log^3 n)$ med småskolemultiplikation. Arbetet kan minskas genom:

- Liten öppen exponent.
- Räkna mod *p* och *q* var för sig. (Detta kan öppna för angrepp!)
- Snabb multiplikation

4.3.4 Euklides' algoritm

Vi utgår från ett exempel. Vi skall finna största gemensamma delaren till 33 och 24 och räknar så här:

$$\begin{aligned} 33 &= 1 \cdot 24 + 9 \\ 24 &= 2 \cdot 9 + 6 \\ 9 &= 1 \cdot 6 + 3 \\ 6 &= 2 \cdot 3 \end{aligned}$$

Läser man uppifrån och ner, så ser vi att varje delare till 33 och 24 delar i tur och ordning 9, 6 och 3. Läser vi däremot nerifrån och upp, så framgår det att 3 delar 6, 9, 24 och 33. Största gemensamma delaren till 33 och 24 är därför 3.

Vi känner på oss, att metoden går på $O(\log n)$ steg, eftersom resterna även i värsta fall nästan halveras varje gång. Därmed blir tiden totalt högst $\log^3 n$. Vid närmare eftertanke ser man dock att $\log^2 n$ räcker, för om kvoterna blir små, blir divisionen blir nästan en subtraktion, och om de är stora, så går det desto fortare.

Fantastiskt att en så gammal algoritmen fortfarande utvecklas och analyseras!

4.3.5 Diverse andra operationer

Gamla och nya metoder och knep har tillämpats på så olika områden som polynomaritmetik, inklusive mod, potensserier av 1/polynom och polynomevaluering i olika talkroppar.

Här kan man också nämna Schimmlers sorteringsalgoritm. Den använder en specialmaskin om $O(m^2)$ kretsar och sorterar m^2 element på $8m$ tid. Detta ger en kostnad på $m^{3+o(1)}$ i stället för $m^{4+o(1)}$ för vanliga sorteringsalgoritmer.

5 Satser om primtal

5.1 Primtalens fördelning

Vi definierar

$\pi(n)$ = antal primtal $\leq n$

Gauss förmodade, Hadamard och de la Vallée Poussin bevisade 1896, att

$$\pi(n) \approx \frac{n}{\ln n}$$

Satsen bevisas med hjälp av analytiska funktioner eller med elementära men långtråkiga uppskattningar. Av resultatet eller beräkningarna följer också att

$$\sum_{\substack{p \leq n \\ primtal}} \frac{1}{p} \approx \ln \ln n$$

vilket utnyttjas i avsnitt 6.1.

5.2 Förväntat antal faktorer

För att uppskatta tid och utrymme för faktoriseringsalgoritmer behöver man veta något om hur faktoriseringar vanligtvis ser ut.

Antalet primfaktorer i n är $\leq \lg n$, och kan nå den gränsen, men genomsnittliga antalet primfaktorer, liksom genomsnittliga antalet olika primfaktorer för tal av storleken n är ln ln n.

Största primfaktorn är ofta $\approx n^{0,63}$ och näst största primfaktorn $n^{0,23}$.

5.3 Glatta tal

Hur vanligt är det med tal med bara små primfaktorer? Vi säger att ett tal x är *glatt* (tidigare *runt*) med avseende på y om alla primfaktorer i x är $\leq y$. Vi definierar också:

$$\psi(x, y) = \text{antal tal} \leq x \text{ vars alla primfaktorer är} \leq y.$$

Då kan man bevisa att:

$$\psi\left(x, x^{1/u}\right) = x \cdot u^{-u+o(1)}$$

Med litet räkningar kan man skriva om detta till:

$$\psi\left(n^{\alpha}, L^{\beta}(n)\right) = n^{\alpha} \cdot L^{-\alpha/(2\beta)+o(1)}(n)$$

där

$$L(n) = \exp\left(\sqrt{\ln n \cdot \ln \ln n}\right)$$

Antydan till bevis. Man kommer en bit på väg genom att se på hur tal i ett visst intervall som man kan skapa med hjälp av ett begränsat antal primtal.

Fantasifyllda uppskattningar med god effektivitet har gjorts av Daniel J. Bernstein, som också har massor med referenser inom området.

5.4 Riemanns hypotes

Euler definierade för komplexa s

$$\zeta(s) = \sum_{1}^{\infty} \frac{1}{n^s}$$

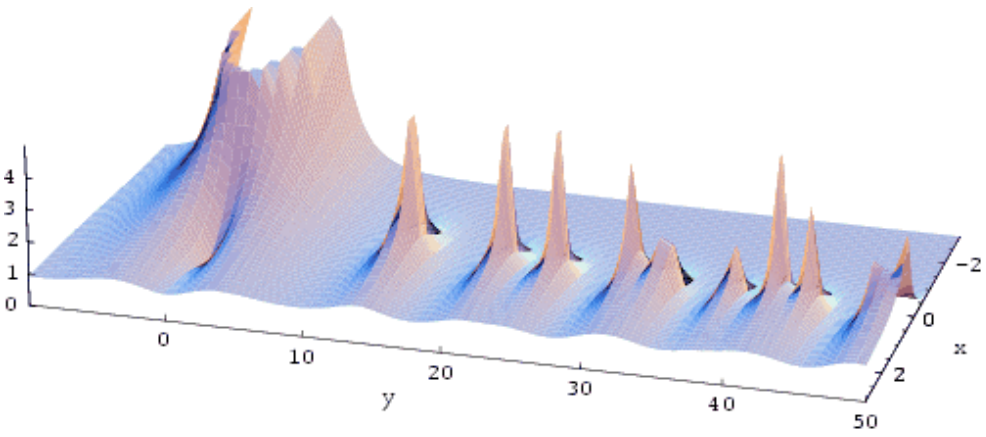
Serien är konvergent om Re(s) > 1, och då är också

$\zeta(s) = \prod_p \frac{1}{1-p^s}$, där produkten tas över alla primtal *p*.

Funktionen kan genom analytisk fortsättning definieras i hela det komplexa planet, bortsett från en pol för *s* = 1. Funktionen har stor betydelse för olika satser om primtal, vilket man gärna tror när man ser den senare formen.

Riemann trodde redan 1859 att

> *Alla nollställen till zeta-funktionen i remsan 0 < Re(s) < 1 ligger på linjen Re(s) = ½.*



Figur. ”Foto” av |1/ζ|. ”Fjälltopparna” markerar nollställen, och de ligger alla på Re(s) = ½. Hämtad från http://www.maths.ex.ac.uk/~mwatkins/zeta/ss-n.htm

Om denna hypotes och en utvidgning av den är sann, så får man många vackra resultat. Utan den får man ofta sämre resultat med mer möda.

Nu har man visat, att 273 miljarder nollställen ligger exakt på denna linje. Detta har skett genom numeriska räkningar: Först räknar man ut antalet nollställen i ett område, och sedan räknar man antalet teckenväxlingar av en funktion på linjen. De numeriska bevisen är alltså synnerligen övertygande, men en spännande osäkerhet kvarstår.

Den som vill faktorisera med stöd av Riemanns utvidgade hypotes har däremot fortfarande, som Pomerance påpekar, ett intressant läge: Om man skulle misslyckas, så har man kanske förlorat litet tid, men i gengäld har man vunnit evig ära genom att ha motbevisat Riemanns förmodan.

För beviset av denna sats finns ett pris på 1 miljon dollar. Inte minst för detta finns en ständig ström av mer eller mindre galna personer som lockas av äran och pengarna.

6 Faktoriseringsmetoder

6.1 Eratostenes' såll

Denna vördnadsvärda algoritm får tjäna som inledning, trots att den inte i första hand är en faktoriseringsmetod.

Uppgiften är att hitta alla primtal ≤ *n*. Vi skriver upp alla tal ≤ *n* i en rad, markerar dem som är > 2 och delbara med 2, ser att det första icke strukna talet är 3, stryker alla efterföljande tal som är delbara med 3 o.s.v. ända till dess att vi kommer till $\sqrt{n}$. De tal som är kvar måste vara primtal, för de är ≤ *n*, men har ingen primfaktor ≤ $\sqrt{n}$.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			*		*		*		*		*		*	
					*			*			*			*

Arbetet blir

$$\sum_{p \le \sqrt{n}} \frac{n}{p} \approx n \cdot \ln\ln n$$

enligt resultatet ovan. Tiden kan pressas (Mairson och Pritchard) till $n/\ln\ln n$. Atkin och Bernstein har klarat samma tid på utrymmet $n^{1/2+o(1)}$ bitar, förutom primtalstabellen.

I resten av avsnitt 6 betecknar *n* det tal som skall faktoriseras.

6.2 Division

Man dividerar med primtal, ett i taget. När man hittat en faktor, provar man den igen, så att inte multipla faktorer kommer bort. Därefter görs primalitetstest innan man fortsätter med nästa primtal. Man är klar, då kvarvarande tal är ett primtal. Detta inträffar senast när primtalen uppnått roten ur den kvarvarande faktorn. Tiden för en fullständig faktorisering beror därför på näst största faktorn i *n*.

6.3 Fermat, Lehmer

Fermat utgick ifrån

$$p \cdot q = n$$

Om man sätter

$$p = x - y$$
$$q = x + y$$

så blir

$$x^2 - y^2 = n$$

$$y^2 = x^2 - n$$

Det sista högerledet beräknas för $x = \lfloor \sqrt{n} \rfloor$ och sedan för följande heltal genom successiva additioner. Blir resultatet en kvadrat, så finns en faktorisering.

Detta går fort om det finns en primfaktor nära $\sqrt{n}$. Annars går det bedrövligt långsamt. I stället kan man prova att faktorisera *kn*. Med systematiskt valda *k* kan man komma ner till $n^{1/3}$ i tid och utrymme.

6.4 Pollard - Strassen

Även denna metod är deterministisk. Den räknar ut *k*! (mod *n*). Med hjälp av snabb evaluering av polynom, snabb multiplikation och snabb modulo av polynom går alltsammans – egendomligt nog – på $O(n^{1/4+\varepsilon})$ tid och lika mycket utrymme. Metoden saknar dock praktiskt intresse.

6.5 Pollards ρ-metod

n skall faktoriseras. Låt *p* vara något primtal som delar *n*. Definiera en pseudoslumptalsföljd med hjälp av en aritmetisk funktion *f* mod *n*:

$$x_{j+1} = f(x_j) \bmod n, \; j = 1, 2, 3 \ldots$$

och den därmed sammanhängande följden

$$y_1 = x_1 \bmod p$$
$$y_{j+1} = f(y_j) \bmod p, \; j = 1, 2, 3 \ldots,$$

I den senare följden kommer man med stor sannolikhet tillbaka till någon punkt y_j inom tiden $c \cdot \sqrt{p}$ (jämför födelsedagsproblemet). Då blir

$$y_{j+k} = y_j, j \geq v$$

För samma *j* och *k* blir

$$\gcd(x_{j+k} - x_j, n) \geq p$$

Periodiciteten upptäcks genom att man räknar ut x_j och x_{2j} samtidigt. Då blir

$y_{2j} = y_j$ för något $j; \nu \le j \le \nu + k$

Detta innebär nästan alltid en faktorisering. Ett enkelt val av *f* är

$$f(x) = x^2 + 1$$

Med småskolearitmetik blir tiden $c \cdot \log^2 n \cdot \sqrt{p}$, där *p* är den minsta primtorn i *n*. Variansen är liten. Utrymmesbehovet är fantastiskt: Linjärt i log *n*. Inte ens en primtalstabell behövs.

6.6 Shanks kedjebråk

Här utvecklas $\sqrt{n}$ i kedjebråk. Man får en mängd kvadratiska rester, som alla är $\le 2\sqrt{n}$. När en sådan rest blir en kvadrat, så har man fått en kongruens

$$x^2 \equiv y^2 \bmod n$$

$$(x - y) \cdot (x + y) \equiv 0 \bmod n$$

som kan ge en faktorisering av *n*.

Det finns $\sqrt[4]{n}$ st. kvadrater $\le \sqrt{n}$. Sannolikheten att få en kvadrat är alltså $\frac{\sqrt[4]{n}}{\sqrt{n}} = \sqrt[4]{n}$ och tiden för att vänta på den är alltså $\sqrt[4]{n}$. Utrymmesbehovet är litet.

Metoden hanterar alltså små tal, som ofta kan lagras i ett ord. Beräkningstiden uppvisar stor varians.

6.7 Pollard *p* – 1

Denna metod fungerar endast om *p* – 1 är glatt, men den är snabb och visar framåt.

Vi vill faktorisera *n*, som har en primfaktor *p*. Låt *a* vara ett tal sådant att gcd(*a*, *n*) = 1.

Om vi successivt skulle räkna ut:

$$y_1 = a^{p_1^{u_1}} \pmod p$$

$$y_2 = y_1^{p_2^{u_2}} = a^{p_1^{u_1} \cdot p_2^{u_2}} \pmod p$$

…

där p_1, p_2, … är en uppräkning av primtalen, så kommer så småningom *p* – 1 att dela produkten i exponenten, och då blir $y_k = 1$. Detta inträffar under förut-

sättning att $p_k \geq q$, där q är den största primtalspotensen i $p - 1$, och att vi har valt exponenterna u_i tillräckligt stora, t.ex. $> \ln q / \ln p_i$.

Vi kan inte räkna ut följden $\{y_i\}$, eftersom vi inte känner p. I stället sätter vi

$$x_1 = a^{p_1^{u_1}} \pmod{n}$$

$$x_2 = x_1^{p_2^{u_2}} = a^{p_1^{u_1} \cdot p_2^{u_2}} \pmod{n}$$

…

och eftersom $x_i \bmod p = y_i$, så blir $\gcd(x_k - 1, n) \geq p$, och vi har nog en faktorisering. Om vi multiplicerar ihop talen $x_i - 1 \pmod{n}$, så behöver inte utföra Euklides' algoritm varje gång.

Vi behöver göra högst $2 \cdot \lg q \cdot \pi(q) = O(q)$ multiplikationer mod n, plus mera sällsynta Euklides' algoritmer.

När bara en primfaktor återstår kan vi förenkla: I första steget räknar vi ut t.o.m. x_l. Sedan beräknar vi differenserna mellan konsekutiva primtal ≥ p_l och x_l upphöjt till dessa differenser. Det blir inte så många. I steg 2 multiplicerar vi den sista potensen x_l med x_{l-1} upphöjt till differenserna, och får därigenom bara en multiplikation per primtal att samla ihop till beräkningar av största gemensamma delare, och får alltså bara 2 multiplikationer i stället för i genomsnitt $1.5 \cdot \lg q$.

Utrymme behövs för att lagra primtal, differenser och potenser. Metoden är snabb, när $p - 1$ bara har små primfaktorer. Detta kan vi tyvärr inte styra, men det går i nästa metod.

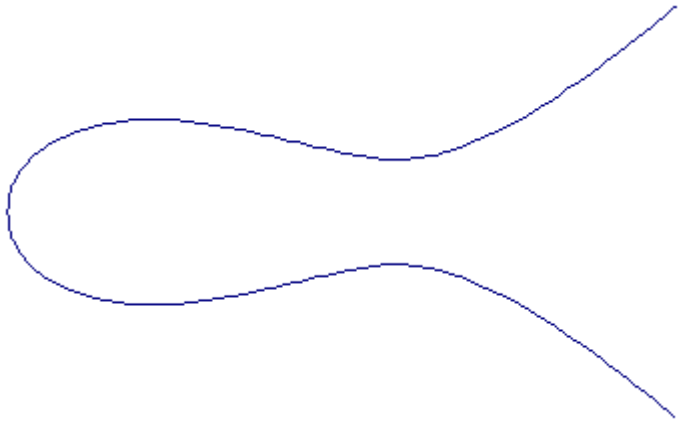
6.8 Elliptiska kurvor

Metoden påminner om Pollard $p - 1$, men använder en annan grupp, som vi nu skall definiera.

Elliptiska kurvor är tredjegradskurvor. De kan bringas till Weierstrass' normalform:

$$\{(x, y), y^2 = x^3 + ax + b\}.$$

I det reella planet kan de se ut så här:



Ofta är det praktiskt att betrakta kurvorna i det projektiva planet. En *z*-koordinat läggs till, ekvationen görs homogen, proportionella triplar är ekvivalenta och en oändlighetspunkt tillkommer.

Man definierar en gruppoperation °: punkterna P_1 och P_2 ger upphov till en ny punkt P_3, som är spegelbilden av skärningen mellan kurvan och linjen genom P_1 och P_2. Är P_1 och P_2 lika, så tar man tangenten i stället, och om P_1 och P_2 är varandras spegelbilder, så blir $P_1 \circ P_2$ oändlighetspunkten, som också är enhetselementet. Inversen är alltså spegelbilden i *x*-axeln.

Koordinaterna till skärningspunkten ges av en tredjegradsekvation, där två rötter är kända. Den tredje roten ges därför av rationella uttryck av de ursprungliga koordinaterna. Vi skriver ner dessa:

Givet två punkter (x_1, y_1) och (x_2, y_2), $x_1 \neq x_2$ så ges sammansättningen av punkterna

$$(x, y) = (x_1, y_1) \circ (x_2, y_2)$$

av

$$x = m^2 - x_1 - x_2$$

$$-y = m(x - x_1) + y_1$$

där

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Man ser att operationen alltid är definierad och att den är kommutativ. Däremot kräver beviset för associativiteten hemska räkningar.

Även om de geometriska tillämpningarna inte syns, så fungerar detta i olika talkroppar, t.ex. i de reella talen, de komplexa talen och i heltalen modulo *p*.

En elliptisk kurva modulo *p* har givetvis ändligt många element. Hur många? Hur ofta blir högerledet en jämn kvadrat? Vi tror gärna att det skall vara i hälften av fallen, och då finns två *y* till varje *x*, så det borde vara ungefär *p* punkter på kurvan.

Men vi kan gissa mer än så. Vi använder Legendre-symbolen

$$(a/p) = \begin{cases} 1 & \text{om a är en kvadratisk rest (mod p)} \\ 0 & \text{om } p \mid a \\ -1 & \text{annars,} \end{cases}$$

Då blir antalet element på kurvan:

$$\#\mathrm{E}(\mathbf{F}_p) = p + 1 + \sum_{x \in F_p} \left((x^3 + ax + b)/p \right)$$

alltså $p + 1 +$ plus ett antal av tal som är +1 eller -1 med lika sannolikhet. (Hälften av alla tal i $\mathbf{F}_p$ är kvadrater!)

Ordningen för en elliptisk kurva borde därför finnas inom ett intervall av längd $\approx \sqrt{p}$. Det visar sig också att ordningen alltid finns i intervallet

$$\left[p + 1 \pm 2\sqrt{p} \right]$$

Fler precisa och för metoden gynnsamma resultat finns.

Tar man ett slumpvis valt gruppelement och beräknar sådana potenser av detta, att exponenterna blir delbara med ett stort antal primtal, så kommer man, om gruppens ordning är glatt, att få fram enhetselementet. I steget dessförinnan är $x_2 - x_1 \equiv 0 \pmod{p}$.

Nu gör vi samma sak mod *n*, där *n* är det tal som vi önskar faktoruppdela. Allt fungerar som vanligt, så länge som

$$\gcd\left((x_2 - x_1), n\right) = 1$$

Är å andra sidan största gemensamma delaren > 1, så har vi (nog) fått en faktorisering. Men i steget har vi uppnått just detta.

För många slumpmässigt valda kurvor (mod *n*) väljer vi ett slumpmässigt element, räknar ut element med hög potens och räknar ut största gemensamma delare mellan *n* och skillnaden i x-koordinater.

En gruppoperation på en elliptisk kurva kan göras på olika sätt. Beräkningarna kan innebära 6 multiplikationer av tal modulo *n*. Till detta kommer en Euklides' algoritm, men denna kan "spädas ut" om man tar flera kurvor på en gång eller undvikas helt med andra parametriseringar av kurvan.

Om vi ger upp tidigt, så får vi undersöka fler kurvor. Om vi håller på länge på varje kurva, så behöver vi inte behandla så många. Vi vet att

P(kurvan glatt med avseende på $L^{\beta}(p)$) = $1/L^{1/2\beta+o(1)}(p)$

Antalet elliptiska operationer för att komma fram till enhetselementet för sådana glatta kurvor blir $L^{\beta}(p)\cdot \ln p$. Arbetet blir alltså

$$O\left(\ln^2 n \cdot \ln p \cdot L^{\beta+1/2\beta}(p)\right).$$

De komplicerade argumenten avslutas så en gymnasieuppgift: minimera

$$f(\beta) = \beta + \frac{1}{2\beta}.$$

f har minimum = $\sqrt{2}$ för $\beta = 1/\sqrt{2}$. Tiden blir alltså

$$O\left(\ln^2 n \cdot \ln p \cdot L^{\sqrt{2}}(p)\right)$$

Det finns en rad möjligheter till optimering. Ett andra steg kan införas som i Pollard $p-1$, varje gruppoperation kan göras med färre multiplikationer, Euklides' algoritm kan undvikas nästan helt, kurvorna kan få ordningar som alltid är delbara med 12, exponentieringen kan göras med färre multiplikationer, och man kan använda snabba multiplikationer.

Detta innebär, att den elliptiska kurvmetoden alltid hittar faktor på högst 10^{20} under en helg med hjälp av en enda PC, att den ofta hittar faktorer på 10^{30} på kraftigare maskiner, medan större faktorer sällan hittas. Rekordet från oktober 2001 ligger dock på 55 siffror, men då har man också haft "tur" med kurvans ordning.

6.9 Kvadratiska sållet

Vi börjar med ett numeriskt exempel (enligt Knuth): Faktoruppdela n = 197209. Vi har på något sätt fått fram kvadratiska rester till n, som också går att dela upp i små primfaktorer:

$159316^2 \equiv 2^4 \cdot 3^2 \cdot 5^1 \bmod 197209$

$133218^2 \equiv 2^0 \cdot 3^4 \cdot 5^1 \bmod 197209$

Då blir

$(159316 \cdot 133218)^2 \equiv 2^4 \cdot 3^6 \cdot 5^2 = (2^2 \cdot 3^3 \cdot 5^1)^2 \bmod 197209$

$126308^2 \equiv 540^2 \bmod 197209$

$(126308 + 540) \cdot (126308 - 540) \equiv 0 \bmod 197209$

gcd(126308 – 540, 197209) = 199

och vi har fått

197209 = 199 · 991.

I ett riktigt fall behöver vi många kvadratiska rester som går att dela upp i små primfaktorer. Ett sätt är att ta polynomet:

$$f(a) = \left(\lfloor \sqrt{n} \rfloor + a\right)^2 \bmod n,\ a = \pm 1, \pm 2,\ \ldots$$

Resterna är till att börja med av storleken $\sqrt{n}$. Men vi vill inte faktorisera för många tal. Därför *sållas* de glatta talen fram.

Vi betraktar en mängd små primtal p_1, p_2, … Vi hittar först ett a_0 sådant att

$$\left(\left(\lfloor \sqrt{n} \rfloor + a_0\right)^2 \bmod n\right) \equiv 0 \mod p_1$$

Vi har valt p_1 så att ekvationen har en lösning, och då går det snabbt att hitta den. $a_0 + p_1, a_0 + 2p_1 \ldots$ uppfyller samma ekvation.

På samma sätt hittar vi rester som är delbara med p_2. Vi sållar ut glatta tal genom att först mycket ungefärligt beräkna logaritmen för polynomets värde. Detta är lätt, eftersom denna funktion är nästan konstant över långa intervall. Sedan markerar man de tal som är delbara med p_1 genom att subtrahera bort $\log p_1$, och på samma sätt med de andra primtalen. När resten blir 0 har vi ett tal som går att faktorisera inom den givna primtalsbasen.

Man bör också sålla med potenser av primtal.

log (*f*) -log p_2	log (*f*) -log p_1	log (*f*)	log (*f*) -log p_1 -log p_2	log (*f*)	log (*f*) -log p_1	log (*f*) -log p_2	log (*f*) -log p_1	log (*f*)
>0	>0	>0	≈0 faktoriseras	>0	>0	>0	>0	>0

Figur. Sållningen sker genom subtraktion av log p_i

När vi har fått tillräckligt många glatta tal, så faktoriserar vi dem. Därefter multiplicerar vi ihop lämpligt valda faktoriseringar med varandra, så att man bara får jämna primtalspotenser. Detta uppnås genom att betrakta exponenterna som vektorer över GF(2), och söka en linjärkombination som blir nollvektorn.

Vi har alltså arbete med att

- Sålla fram glatta tal
- Faktorisera dessa

- Lösa ett linjärt binärt ekvationssystem
- Multiplicera ihop talen igen.

Vi tar B element i primtalsbasen. Vi skall alltså sålla fram B stycken tal av storlek $\sqrt{n}$ som är glatta med avseende på B. Som vi såg vid Eratostenes' såll var arbetet per sållat tal = ln ln B, vilket är litet i sammanhanget. Sätter vi nu $B = L^{\beta}(n)$, så får vi

$$\mathbf{P}(\text{glatt}) = L^{-1/4\beta}(n)$$

Antal glatta tal som behövs är $L^{\beta}(n)$.

Arbetet med sållningen blir alltså

$$L^{\beta+1/4\beta}(n)$$

Vi får minimum $L(n)$ för $\beta = ½$. Faktoriseringen tar också $L(n)$ tid.

Utrymmet för sållningen skulle också bli $L(n)$, men vi kan dela upp sållningsintervallet i delar. Lagrar vi alla element i bitmatrisen, så tar den $L(n)$ bitar, vilket kan tvinga oss att ta B litet mindre än optimalt.

Arbetet att lösa ekvationssystemet med vanliga metoder är ungefär B^3, men det sker med operationer som kan göras parallellt i maskinens ord. För större B väljer man metoder för glesa matriser som tar B^2 tid. Matriserna är verkligen glesa, för talen har ju högst $\ln\sqrt{n}$ olika primfaktorer. Tiden för ekvationslösningen blir därför av samma ordning som tiden för sållningen.

Tiden $O\left(L^{\sqrt{2}}(p)\right)$ för elliptiska kurvor med maximalt ofördelaktiga $p = \sqrt{n}$ blir $L(n)$, vilket är samma som för kvadratiska sållet. Men de elliptiska kurvorna behöver för varje gruppoperation en handfull multiplikationer modulo n, medan det kvadratiska sållets vanligaste åtgärd bara är en ynka addition av tal med liten precision. Dessutom finns många förbättringar att göra.

Elliptiska kurvor är överlägsna för att få fram "små" faktorer, medan kvadratiska sållet är bäst för riktigt svåra faktoriseringar. Ett gammalt rekord (1994) ligger på 129 siffror.

6.10 Gauss' metod

Som en historisk kuriositet och en antydan om hur mycket tankeverksamhet som ligger bakom dagens metoder, så skall också Gauss' metod nämnas.

Metoden bygger på att om $p \mid n$, så

$$a \equiv x^2 \bmod n \Rightarrow a \equiv x^2 \bmod p$$

Som delare till n duger bara sådana p som uppfyller

$$(a / p) = 1$$

för alla kvadratiska rester a. Antalet tänkbara p halveras för varje a vi provar. För att hoppas på ett någorlunda unikt p måste vi alltså ha cirka lg p stycken rester. Resterna skall vara små för att beräkningarna skall gå fort.

Även denna metod samlar alltså på glatta kvadratiska rester. De kombineras med hjälp av linjära ekvationssystem modulo 2, så att vi får fram primtal som är kvadratiska rester. Sedan används Gauss' djupa sats om den kvadratiska reciprociteten för att räkna ut (a / p). Har vi plockat ut tillräckligt många a, så kan vi förmoda att det största a vi behöver är $O(\ln p)$.

De magnifika förberedelserna, helt i klass med det tyngsta arbetet med det kvadratiska sållet, ger tyvärr ringa utdelning. Vi måste prova alla primtal $\leq \sqrt{n}$ och räkna ut (a / p), vilket går ungefär lika fort som att räkna ut gcd(a, p). I genomsnitt måste detta test göras två gånger. Detta är förvisso snabbare än att dividera n med p, vilket kräver $\log p \cdot \log n$ operationer, men det tar ändå $\log p \cdot \log \log p$ tid.

Gauss vann alltså en avsevärd förbättring gentemot enkel division, men 200 års forskning passerar även historiens största matematiker.

6.11 Talkroppssållet

Länge var tiden *O(L(n))* en oöverkomlig gräns. Flera metoder byggde på förekomsten av glatta tal av storleksordningen $\sqrt{n}$. I *talkroppssållet* från början av 90-talet sållar man bland mindre tal och får en asymptotiskt snabbare algoritm.

I gengäld måste man också faktorisera algebraiska heltal, som sägs vara glatta om deras *norm* är glatt. Metoden bygger på räkningar i Galoiskroppar, men i övrigt liknar den det kvadratiska sållet.

Programmeringen är mycket komplicerad. Den kräver stora bakgrundskunskaper. Många problem måste lösas. Flera nya moment kräver uppmärksamhet och optimering.

Mot den bakgrunden är det imponerande att den första implementationen gjordes på en 8 bits persondator med mycket begränsat diskettutrymme. Vill man så kan man!

Komplexiteten ges av det underbara uttrycket

$$\exp\left(\left((64/9)^{1/3} + o(1)\right)\cdot (\ln n)^{1/3} \cdot (\ln \ln n)^{2/3}\right).$$

Detta är asymptotiskt bättre än $L(n)$. Båda uttrycken är mellanting mellan uttryck som växer exponentiellt eller polynominellt med ln n. Talkroppssållet är snabbare om

$$\frac{(64/9)^{1/3} \cdot (\ln n)^{1/3} \cdot (\ln \ln n)^{2/3}}{(\ln n)^{1/2} \cdot (\ln \ln n)^{1/2}} = \left(\frac{4096 \cdot \ln \ln n}{81 \cdot \ln n}\right)^{1/6} < 1$$

eller

$$\frac{\ln\ln n}{\ln n} < \frac{81}{4096}$$

vilket enkelt löses numeriskt och ger $n > 10^{125}$. Exakt var punkten för break-even mellan det kvadratiska sållet och talkroppssållet beror givetvis på respektive implementation och på hur särskilt talkroppssållet förbättras. Siffran 10^{105} har nämnts.

Detta är alltså inte lönsamt för vanliga persondatorer, eftersom $L(10^{125})= 3\cdot10^{17}$, och man f.n. knappast kan göra mer än 10^{11} multiplikationer under en helg. Rekordet ligger på 158 siffror, och sållningen tog ett par månader.

Daniel. J. Bernstein har gjort viktiga iakttagelser beträffande talkroppsållet. Sållningen ser ut att vara snabb, bara ln ln *n* operationer per tal, men den utnyttjar inte minnet optimalt: Bitarna sitter och rullar tummarna medan de väntar på att bli strukna! Ett glatthetstest går långsammare än sållning, men fortfarande på o(*n*) tid och bara *O*(ln *n*) minne. Det blir billigare. Vidare kan man med hjälp av Schimmlers sorteringsalgoritm multiplicera en gles kvadratisk binär matris med en vektor av längden *m* på $m^{0,5+o(1)}$ tid i stället det vanliga $m^{1+o(1)}$.

Om man bygger specialmaskiner för dessa algoritmer, så skulle man till samma kostnad kunna klara tal med tre gånger fler siffror jämfört med talkroppssållet på en vanlig dator. Detta skulle innebära ett genombrott.

Kvadratiska sållet kan optimeras på samma sätt. Jag tror att det skulle bli drygt man dubbelt så många siffror som på en generell dator.

6.12 Övriga allmänna metoder

Det finns många olika källor till matematiska faktoriseringsmetoder. En av dem är *klassgrupper*, som redan Gauss sysslade med.

Man utgår från kvadratiska former i två variabler med heltalskoefficienter, definierar en ekvivalensrelation (transformation med heltalsmatriser med determinant 1), definierar kanoniska element och ger algoritm för att nå dit. Slutligen definierar man en gruppoperation på former med samma diskriminant. Dessa grupper, som alltid finns och vars ordningar ligger nära diskriminanten, är ganska lätta att beräkna. Vidare hänger element av ordningen 2 nära ihop med faktorisering av diskriminanten.

Klassgrupperna kan användas på olika sätt: *Klassgruppsmetoden* använder dem som elliptiska kurvor och väljer klassgrupper med diskriminanter som är små multipler av *n*. Därmed får man grupper av slumpmässig ordning, varav några förhoppningsvis är glatta. Tiden blir *L(n)* och utrymmet linjärt i ln *n*. Seysens klassgruppsalgoritm faktoriserar de kvadratiska formerna och kombinerar dem på ett sätt som påminner om kvadratiska sållet och ger samma asymptotiska komplexitet, men med mycket högre konstanter.

6.13 Speciella metoder

Om man arbetar med speciella typer av tal, t.ex. Mersennes eller Fermats tal (se avsnitt 8.1.2), så finns anpassningar av de allmänna metoderna och dessutom speciella metoder. Det skulle föra för långt att här gå igenom detta.

7 Karaktärisering av metoder

7.1 Bevisläge

Vissa metoder är bevisat *deterministiska* och ger alltid en äkta faktorisering inom den tid som garanteras. Andra är deterministiska, men *bygger på vissa förmodanden*. Mest använd är den utvidgade Riemann-hypotesen. Andra antaganden kan gälla att det finns tillräckligt många primtal, glatta tal eller dylikt, i det urval man betraktar.

Övriga metoder är *probabilistiska* och använder slumpvis valda tal, som leder fram till en faktorisering. Av dessa metoder har somliga en *bevisbar sannolikhet* att lyckas. I andra fall bygger bevisen på någon känd *förmodan*, t.ex. den utvidgade Riemann-hypotesen. Resten av metoderna är *heuristiska*. Här är antagandena mer osäkra. Tidsuppskattningen för det kvadratiska sållet bygger på att andelen glatta tal bland vissa kvadratiska rester är lika stor som bland slumpvis valda tal av samma storleksordning.

Talteorin är fascinerande eftersom man ofta med elementära kunskaper kan gissa sig fram till djupa resultat. Gissningarna är ofta riktiga och kräver mindre kunskaper än strikta bevis.

De probabilistiska metoderna fungerar nästan alltid och tar mindre förväntad tid. Som synes nedan får man betala ett högt pris för att vara helt säker på sin sak.

Men visst bör man eftersträva deterministiska metoder, eller i varje fall metoder med bevisbar sannolikhet. Vetenskapens värdighet kräver detta.

7.2 Förväntad beräkningstid

Nedan står p för det minsta primtalet i n och q för största primfaktorn i p. Uppskattningarna gäller klassisk multiplikation.

Bevisbarhet och metod	**Förväntad tid = *O( )***
Deterministiska	
Pollard – Strassen	$n^{1/4} \cdot \ln^2 n$
Pollard $p - 1$	$q \cdot \ln^2 n$
Lehmer	$n^{1/3} \cdot \ln^2 n$
Division	$p \cdot \ln n$
Deterministiska under förmodan	

Klassgruppsmetod	$n^{1/5} \cdot \ln^3 n$
Probabilistiska	
Klassgruppsrelationer	$L(n)$
Variant av kvadratiska sållet	$L^{\sqrt{4/3}}(n)$
Elliptiska kurvor (nästan probabilistisk)	$L^{\sqrt{2}}(p)$
Probabilistiska under förmodan	
Annan klassgruppsrelation	$L(n)$
Heuristiska	
Talkroppssållet	$\exp\left(c \cdot (\ln n)^{1/3} \cdot (\ln \ln n)^{2/3}\right)$
Kvadratiska sållet	$L(n)$
Pollard ρ	$p^{1/2} \cdot \ln^2 n$
Shanks kedjebråk	$n^{1/4}$
Gauss	$L(n) + p \cdot \ln \ln p$

7.3 Varians av beräkningstid

7.3.1 Deterministiskt, beroende av talet

Några deterministiska metoder är kraftigt beroende av det tal man skall faktorisera.

Hit hör division, vars tid beror på näst största primfaktorn, Fermat, som går fortast om det finns en faktor nära $\sqrt{n}$, och Pollard $p - 1$ som beror på och den största faktorn i $p - 1$, p är minsta primfaktorn i n.

Beräkningstiden beror alltså på om talet n är lätt eller svårt.

7.3.2 Kraftig varians

Andra metoder har mycket stor spridning av skäl som hör samman med algoritmen. Den elliptiska kurvmetoden väntar på *en* kurva med glatt ordning, och om sannolikheten för detta är $1/t$, så blir väntetiden t och spridningen t, vilket är mycket. Samma gäller för Shanks kedjebråk, där man väntar på en enda kvadrat.

7.3.3 Måttlig varians

I andra fall visar enkla statistiska modeller på måttlig varians.

I Pollard ρ väntar man på att två punkter skall sammanfalla. Detta blir alltmer sannolikt ju längre tiden går, vilket minskar variansen av beräkningstiden för tal vars minsta primfaktor har en viss storleksordning.

Det kvadratiska sållet väntar på ett stort antal glatta tal, och variansen blir därför måttlig.

Måttlig varians är behaglig för programmeraren, som kan se när hans optimeringsförsök ger effekt.

7.4 Ändamål

Somliga metoder hittar i första hand små faktorer. Hit hör division, Pollard ρ, Pollard $p - 1$ och elliptiska kurvor.

7.5 Minneskrav

Några metoder har mycket små minneskrav: linjära i log n. Hit hör Pollard ρ, som i princip endast använder ett par tal av storlek n.

Några andra metoder har linjärt utrymme, men förutsätter att det finns en tabell med primtal och primtalspotenser. Hit hör elliptiska kurvor, Pollard $p - 1$, division och en klassgruppsalgoritm. För de två första kan man i steg 2 minska utrymmet från B_1 till $\sqrt{B_1}$, där B_1 är gränsen för primtalen i steg 2. Detta spelar inte så stor roll, eftersom denna tabell är gemensam för flera metoder.

Kvadratiska sållet kräver mycket utrymme för olika areor. Låt B vara primtalsbasens storlek. Då tar:

Primtalstabellen	B
Sållningsområdet	$L(n)$, som dock kan delas upp
De faktoriserade talen	$B \cdot \ln^2 n$
Bitmatrisen	B^2, alternativt $B \cdot \ln^2 n$ bitar

Med $B = L^{1/2}(n)$ ser man, att bitmatrisen kan bli problematisk och att metoden kräver effektiv minneshantering.

Slutligen har några algoritmer exponentiellt utrymme, t.ex. Pollard – Strassen med $n^{1/4}$.

7.6 Parallellism

Slutligen kan algoritmerna karaktäriseras efter hur de går att exekvera parallellt.

Pollard ρ, Pollard $p - 1$ och Shank är helt *sekventiella* och ingen parallellism är naturlig.

Schimmlers sorteringsalgoritm utförs parallellt men kräver *extremt snabb koppling* mellan de jämförande enheterna.

Lösningen av de linjära, binära ekvationssystemen i kvadratiska sållet och talkroppssållet kan lösas *parallellt* med hjälp av *tätt kopplade datorer*.

Sållningen i dessa metoder kan göras *parallellt*, och de sållade talen kan lämnas vidare till en *central efterbehandling*. Ingen av maskinerna ute i världen kommer att hitta någon faktorisering på egen hand.

Den elliptiska kurvmetoden fungerar däremot *helt parallellt* och en faktorisering kan upptäckas av vilken maskin som helst.

8 Primtalsindustri och primtalsmästerskap

Att vara duktig på primtal innebär inte bara att komma på en bra metod. Metoden skall dessutom implementeras i program med höga prestanda, i daglig produktion eller i hårda tävlingar. Vi skall nämna ett antal grenar som forskarna tävlar i.

8.1 Faktorisera

8.1.1 Många tal

En uppgift är att faktorisera så *många* tal som möjligt av en viss storlek under en viss tid. Användningen kan vara en subrutin i andra faktoriseringsrutiner, statistik över faktorer i ett visst område eller demonstrationsändamål.

8.1.2 Svåra tal

Ett annat mål är att klara av riktigt *svåra* tal på så kort tid som möjligt. Sådana tävlingar utlyses ibland.

Det kan gälla en RSA-nyckel, som ju är produkten av två primtal utan särskilda egenskaper, eller slå ett världsrekord, eller att klara av resten av en svår faktorisering.

Man blir inte trodd om man presenterar ett tal av slumpmässigt utseende och uppger att det är produkten av två primtal. Då kan man ha utgått från primtalen och multiplicerat ihop dem själv.

8.1.3 Speciella tal

I stället brukar primtalsentusiasterna angripa tal av ett *visst utseende*, givna av Gud eller någon stor matematiker. Vanliga exempel är Mersennes tal $2^n - 1$, eller Fermats tal $2^{2^n} + 1$.

Ett bestämt utseende kan underlätta. Delare till både Mersennes och Fermats tal har speciella egenskaper. Till många Mersenne-tal kan man dessutom hitta algebraiska faktorer med hjälp av nästan enbart gymnasiekunskaper. För att slå rekord med sådana tal behövs fler hjälpmedel än enbart en generell faktoriseringsalgoritm.

8.2 Bevisa primalitet

8.2.1 Hitta användbara primtal

För att använda RSA-metoden måste man skapa stora primtal utgående från slumptal.

Primtalssatsen säger att det går snabbt att hitta primtal. I genomsnitt tar det bara ln *r* försök att hitta ett primtal i närheten av *r*. Bäst använder man

Eratostenes' såll för att hitta kandidater till primtal nära *r*, innan man gör primalitetstester.

Om primtal skall duga som RSA-nyckar finns vissa restriktioner. T.ex. får varken $p - 1$ eller $p + 1$ vara glatta. Man kan ta hänsyn till detta vid lottningen. Detta var viktigare innan ECM-metoden fanns. Nu måste konstruktören räkna med risken att angriparen har tur, men denna risk är försumbar om primtalen är tillräckligt stora.

8.2.2 Bevisa primalitet för RSA-nycklar

RSA-nycklar måste vara primtal. I annat fall får man inte tillbaka klartexten när man dekrypterar.

Man kan nöja sig med att testa med Fermats lilla sats. I första omgången måste ett primtal p uppfylla:

$$2^{p-1} \equiv 1 (\mathrm{mod}\ p)$$
$$3^{p-1} \equiv 1 (\mathrm{mod}\ p)$$

o.s.v. Det finns dock sammansatta tal som klarar detta test för alla baser.

Metoden kan byggas ut. Om den utökade Riemannhypotesen är sann så kan man få bättre tider deterministiskt på polynominell tid. I augusti 2002 presenterades en metod enligt dessa enkla riktlinjer som utan att bygga på några förmodanden bevisade primalitet på polynominell tid. Tyvärr var den odräglig långsam, i första omgången $O(\ln^{19} n)$ med naiv och $O(\ln^{12} n)$ med snabb aritmetik för tal och polynom.

Probabilistiskt med hjälp av elliptiska kurvor kan man på helt acceptabla tider bevisa primalitet för tal med flera tusen siffror.

Frågan är alltså löst. Program som visar primalitet finns tillgängliga på nätet.

8.2.3 Hitta världens största primtal

De största kända primtalen har nästan alltid varit Mersennetal. Speciella tester finns. Avancerad matematik måste användas.

Rekordet från november 2001, $2^{13466917} - 1$, har över 4 millioner siffror.

8.3 Övriga grenar

8.3.1 Räkna antal

En annan gren är att räkna antalet, upp till en viss gräns, av t.ex. primtal eller *primtalstvillingar*, par där såväl p som $p + 2$ är primtal eller glatta tal.

Syftet kan vara att verifiera eller motbevisa en sats eller kanske bara verifiera att en restterm har en viss storlek. I många fall har man nått häpnadsväckande resultat.

Primtalstvillingar har förvisso ingenting att göra med vare sig faktorisering eller samhällets säkerhet, men under sitt arbete att räkna dem upptäckte

Thomas Nicely Pentium-buggen. Felet inträffade under oklara och sällsynta omständigheter, då den minskade noggrannheten till 9 signifikanta siffror. Misstankar kunde riktas mot den egna programvaran, kompilatorn, databussen, tillverkaren och slutligen processorn och dess enhet för flytande aritmetik. Det tog Thomas Nicely tiden från juni till oktober att isolera felet till processorns del för flytande räkning. Efter ytterligare några veckor fick han ett utbyteschips. Sent i december beslöt Intel att ersätta chips för alla som bad om det. Till saken hör, att Intel själva hade upptäckt felet i maj men hållit tyst om det i ett halvår.

För varje programmerare är ett sådant fel en mardröm. Det annars verklighetsfrämmande arbetet med primtalstvillingar ledde alltså till att världen befriades från en felaktig multiplikationstabell, som hemlighölls av ett globalt företag. Talteorin gagnade en rad specialister inom skilda områden.

8.3.2 Numeriska bevis

Redan i avsnitt 5.4 redogjordes för räkningar som gjorde Riemanns hypotes trolig. Just denna gren kan alltså ha förlorat sitt intresse, men andra förmodanden återstår.

8.3.3 Primtalsmästares villkor

Amatörer har ingen chans att slå rekord, vare sig det gäller faktorisera stora slumpmässiga tal, hitta primala Mersennetal, faktorisera Fermat-tal, komma med numeriska argument för Riemanns hypotes eller primtalstvillingars fördelning. Allt måste samverka: Teoretisk expertis på en rad områden, optimeringar på varje punkt – även om det bara ger en faktor 1,5 – de kraftfullaste arbetsstationer som finns att få, snabba interna nät, effektiv programvara, professionell distribution av programvaran, samverkan inom den egna institutionen eller över hela Internet. Bysnillenas tid är förbi.

Projektdeltagarnas villkor beror på algoritmen. En del algoritmer behöver nästan inget samarbete alls. Vem som helst av deltagarna kan ha tur och hitta en faktor, t.ex. med hjälp av elliptiska kurvor. I så fall kan han ta äran av projektledaren, som stått för programutvecklingen, vilket, som i fallet stora Mersennetal, ibland kan ta sig lustiga uttryck.

Det kvadratiska sållet ger projektledaren en trevligare situation. Deltagarna bara får vissa områden att sålla bland och får leverera glatta tal, faktoriserade i primtalsbasen. De har ingen chans att själva upptäcka en faktor. Den centrala datorn finner den slutliga lösningen och projektledaren får äran.

9 Att skriva ett faktoriseringsprogram

9.1 Behovet

Få människor behöver faktoriseringsprogram. Vill man ha ett, så går det att ladda ner från nätet. En amatör har ingen chans mot proffsen.

Men om man är verkligt intresserad, så vill man skriva själv. Det är det roligt att tvingas ta ställning till alla detaljer och lösa ett antal problem, som troligen inte dokumenterats i den litteratur man har tillgänglig. Programmeringen är mer varierad, optimeringen mer komplicerad och prestanda mer förutsebara än i de flesta andra uppgifter. Faktorerna trillar ut som betyg på ens kunskaper. Man vet vilken metod som har använts, man vet hur fort det har gått, man kan prova alternativ. Det är mycket tillfredsställande.

9.2 Nödvändiga kunskaper

9.2.1 Bas

Man behöver inte kunna så mycket. Litet algebra och grundläggande talteori underlättar. I nödfall kan man skriva av formlerna i den bok man läser utan att egentligen förstå något.

Det grundläggande man bör känna till är:

- Euklides' algoritm
- Fermats lilla sats
- Primtalens fördelning
- Fördelningen av glatta tal
- Antal faktorer i slumpmässiga tal
- Fördelningen av största faktor, näst största faktor, … i slumpmässiga tal
- Paket för lång (men inte nödvändigtvis snabb) aritmetik

Vet man detta, så kan man göra prestandauppskattningar.

Med stark tro och god vilja kan man sedan skriva av några enkla metoder och få resultat.

Sedan räcker det med t.ex. Pollard ρ och Shanks kedjebråk. Man har inga minnesproblem, man behöver inte ens en primtalstabell, och man klarar av tal upp till 25 siffror.

9.2.2 Roligt att veta

Man förstår mycket mer om man i varje fall ytligt känner till

- Kvadratiska reciprocitetssatsen
- Elliptiska kurvor

Då gör man en primtalstabell med hjälp av Eratostenes' såll och implementerar Pollard $p - 1$, elliptiska kurvor och det kvadratiska sållet.

9.2.3 Mer att lära

För att ha riktigt roligt bör man kunna bevis inom ovanstående områden och dessutom:

- Analytisk talteori
- Galoisteori
- Algebraiska heltal och deras faktorisering
- Klassgrupper
- …

Om man vill konstruera en ny metod, så måste kunskaperna vara mycket aktiva, för konkurrensen är stenhård.

9.3 Val av metoder

9.3.1 Grundbehov

I första hand tar man det bästa och senaste. Gamla metoder kan förpassas till historiens skräpkammare!

Riktigt så fungerar det inte. Enkelhet är viktigt. I vissa intervall är gamla metoder optimala. Talkroppssållet är svårt att begripa och programmera.

För att med goda prestanda klara tal upp till 60 – 70 siffror räcker:

- Elliptiska kurvor
- Kvadratiskt såll

9.3.2 Snabbhet i specialfall

Vill man ha ett all-round-program så bör man ta med fler metoder.

- *Division* är snabbare än primalitetstest och snabbare än Pollard ρ för små faktorer. Den får lättare bort närliggande faktorer.
- *Pollard ρ* är snabbare än elliptiska kurvor för faktorer med högst 15 siffror.
- *Pollard p* – 1 hittar ofta, men inte alltid, småfaktorer fortare än Pollard ρ
- *Shanks kedjebråk* erbjuder en snabb reservutväg för vilka tal som helst med högst 21 siffror, om t.ex. Pollard ρ har upptäckt flera faktorer på en gång.
- *Rotutdragning* med hjälp av Newton-Raphsons iterationer upptäcker om talet är en jämn potens, vilket t.ex. det kvadratiska sållet inte klarar av.
- *Algebraisk faktorisering* av Mersennetal hittar ofta stora faktorer, vilket kan vara helt avgörande för framgången.
- *Talkroppssållet* måste användas för mycket stora tal
- Någon *deterministisk metod* kan vara rolig att ha i yttersta nödfall. Men jag är tveksam. Man kan inte lyckas jämt. Det finns oändligt många tal som inte är faktoriserade. Varför skall man ha absoluta garantier för att alltid klara ganska små tal?

9.3.3 Primalitetstest

Man kan nöja sig med att de faktorer man får fram nog är primtal. I så fall räcker det med enkla test, baserade på Fermats lilla sats och litet till.

Vill man vara säker, får man ta med ett program för primalitetstester.

9.4 Maskinvara

Man måste bestämma ambitionen för projektet.

Räcker nattkörningar på den egna arbetsstationen? Eller skall man köra på flera maskiner och anpassa programmen till detta? Är målen ännu högre? Vill man köra på all tillgänglig ledig tid inom sin egen institution, eller skall man tigga om fri processorkraft på Internet? I så fall måste beräkningarna kunna delas upp i grova delar. Program och data kunna distribueras och resultaten samlas in. Eller vill man rentav bygga en egen maskin för ändamålet?

Ingenjörer klarar detta. Det går att designa och testa kretsar med samma teknik som när man skriver program. Forskningsinstitutioner kan själva sätta ihop superdatorer med hjälp av standardkretsar.

Förr kunde satsning på speciell hårdvara anses oklok, för om 18 månader brukar man ha tillgång till dubbelt så snabba datorer, och snart är alla specialinvesteringar i maskin- och programvara bortkastade. Tiden urholkar snabbt en konstant faktor. En specialmaskin måste designas för vissa algoritmer och vara mycket bättre för dem.

9.5 Programmeringsteknik

9.5.1 Största tänkbara tal

Man bör tidigt sätta målet för vilka tal man vill klara av. Man får studera metoderna och ta hänsyn till CPU-frekvens, primärminne och sekundärminne.

9.5.2 Språk och operativsystem

Minneshanteringen måste vara effektiv, vilket ställer krav på operativsystem och programspråk.

Assembler kan behöva utnyttjas begränsad omfattning.

9.5.3 Implementeringsdatorns speciella egenskaper

Det finns en rad egenskaper hos datorerna som man bör känna till på ett tidigt stadium i programmeringen. Hit hör:

- Ordlängd
- Begränsad storlek på minnesareor, t.ex. 64-K:s gräns
- Storlek på cache, primär- och sekundärminne
- Operativsystem
- Minneshantering

9.5.4 Allokeringar och friställningar

Det gäller att så tidigt som möjligt uppskatta utrymmeskraven för de olika algoritmerna. Hur stora är de gemensamma tabellerna över primtal och primtalspotenser? Måste programmet självt allokera och friställa utrymmen, eller klarar operativsystemet all garbage collection?

9.5.5 Krav på huvudprogram

Varje metod passar bäst i ett visst intervall.

Exempelvis förbättras det kvadratiska sållets prestanda kraftigt, om faktorer på tillsammans 10 siffror divideras bort. För $n = 10^{50}$ rör det sig om en faktor 100, och för $n = 10^{100}$ blir det i varje fall en faktor 30. Till detta kommer att faktorn $\ln^2 n$, som förekommer på andra ställen i algoritmen, reduceras något. Man måste därför alltid börja med att hitta små faktorer.

Kvadratiska sållet kräver mycket administration. Det bör därför undvikas för små tal.

Huvudprogrammet skall avgöra vilken metod som skall användas och hur länge. Det skall kalla på rutiner för respektive metod. Sådana rutiner skall sätta sina egna parametrar, lämna ifrån sig en faktorer på ett enhetligt sätt och ge eventuella delresultat.

Huvudprogrammet skall presentera samtliga faktorer i lämplig ordning. Det skall också ta hand om fel- och undantagssituationer.

9.5.6 Huvudprogrammets förlopp

- Dividera bort småfaktorer
- Applicera en eller flera småfaktormetoder
- Välj metod med hänsyn till kvarvarande tals storlek
- För tal i lämpligt intervall, t.ex. $10^{20} – 10^{100}$, använd kvadratiska sållet
- (För större tal, kör talkroppssållet)

Desssutom skall man så fort en faktor har hittats:

- Testa att den är primtal, om detta inte redan framgår
- Dividera bort den till dess inga potenser av den finns kvar
- Testa om kvarvarande faktor är primtal

Programmet är klart, när samtliga faktorer är primtal. Då skall det:

- Sortera faktorer, presentera resultat

9.5.7 Specialfall

När småfaktorsmetoderna – Pollard ρ, Pollard $p – 1$ eller elliptiska kurvor – har hittat en faktor, så finns användbar information för fortsättningen. Det är starta om med de gamla resultaten modulo n / p.

Eftersom alla använda metoder utom division är heuristiska eller probabilistiska, och man dessutom i vissa fall gått genvägar, så kan fel inträffa.

I Pollard ρ räknar man inte ut största gemensamma delare varje gång. Det inträffar därför ofta, att två faktorer hittas på samma gång. Då får man backa till tidigare Euklides-beräkning och gå fram med kortare steg.

I såväl Pollard och elliptiska kurvor skulle man kunna hitta två faktorer på samma gång, och att det var de sista faktorerna. I så fall kan man försöka med nya slumptal. Ett annat sätt är att prova med kvadratiska sållet eller Shank, beroende på talets storlek.

Pollard $p - 1$ kan ta orimligt lång tid, om $p - 1$ inte är glatt. Man måste därför avbryta metoden efter ett i förväg angivet antal försök.

I Shanks metod skulle perioden i kedjebråksutvecklingen kunna vara så kort att man aldrig hittar någon jämn kvadrat bland resterna.

Kvadratiska sållet ger enligt min erfarenhet, tyvärr utan riktig förståelse, inga användbara linjära samband om n är en jämn potens. Man behöver därför testa om $n = m^q$. Det räcker om q är ett litet primtal, om vi redan har uteslutit små faktorer i n.

Av dessa skäl, samt för att inte köra för länge med asymptotiskt sämre algoritmer, måste man för varje metod sätta en tidsgräns för användningen.

9.6 Programoptimering

När programmet börjar fungera, är det dags för optimering.

9.6.1 Funktioner

Aritmetik utöver datorn klarar i en instruktion är nödvändig. Skall aritmetiken också vara snabb? Finns paket tillgängliga? Kan dessa användas ihop med det egna programspråket?

Skall man optimera Euklides' algoritm? Skall man ha snabba elliptiska gruppoperationer? Skall man använda snabba metoder för att lösa de linjära ekvationssystemen?

9.6.2 Mätningar

CPU-tid och minnesutrymme skall mätas. Detta kräver en väldefinierad miljö. Mätningarna skall omfatta såväl delmoment – t.ex. multiplikation och Euklides' algoritm – delar av algoritmer, exempelvis sållning och ekvationslösning i kvadratiska sållet, som hela faktoriseringar.

Mätningar måste göras både före och efter försöken till förbättringar. De skall protokollföras. Hela tiden jämför man med teoretiska formler.

9.6.3 Parametersättning

Olika parametrar måste sättas till de olika rutinerna. Det gäller gränserna för steg 1 och steg 2 i elliptiska kurvor och Pollard $p - 1$. För elliptiska kurvor måste man bestämma hur långt man skall testa för varje kurva och hur många kurvor man skall testa, samt antalet parallella elliptiska kurvor (om man

behöver sådana för att undvika Euklides' algoritm). För det kvadratiska sållet måste man bestämma storleken av primtalsbasen, gränsen för stora primtal, tid mellan byten av kurvor och mycket annat.

Gränser måste sättas för när de olika metoderna skall användas. Detta kan göras teoretiskt och praktiskt. Formler för glatta tal visar på att gränsen mellan Pollard ρ och elliptiska kurvor ligger vid c:a 10^{15}.

Snabba maskiner gör arbetet roligare. De kanske inte minskar väntetiden, för denna beror mest på försöksplaneringen, och aptiten växer medan man äter. Men har man en alltför dålig maskin, så är det ingen idé att prova avancerade metoder. I min förra maskin hittade den elliptiska kurvmetoden aldrig någon faktor som inte upptäckts snabbare på annat sätt.

9.6.4 Hur investera?

När alla kalibreringar är klara kommer man fram till konkreta frågor. Vilken blir effekten på befintliga program av:

- Snabbare inre loopar
- Snabbare CPU
- Mer minne
- Större hårddisk?

I mitt projekt fick jag fundera: Skall jag satsa på uppgraderingar av programvaran? Skall jag genomföra önskvärda förändringar? Eller skall jag satsa på bättre maskinvara? Hur många fler siffror klarar jag av med en snabbare CPU?

10 Forcering av RSA-metoden

10.1 Flera angreppssätt

Den hemliga nyckeln till RSA-algoritmen går att hitta, om man kan faktorisera *n*. Detta framgick av avsnitt 3.2. Nya algoritmer för faktorisering är alltså en fara för metodens säkerhet.

Men det är inte den enda faran. Om man har tillgång till det chip som utför krypteringen, så kan mycket inträffa. Man kan lysa igenom kretsarna med röntgenstrålning och läsa nycklarna. Man kan mäta den tid som krävs för exponentieringen och på så sätt få en uppfattning om antalet ettor i den binära framställningen av den hemliga nyckeln.

Algoritmen kan implementeras så, att beräkningarna modulo *p* och *q* sker var för sig. En mera raffinerad forcering utnyttjar detta: genom radioaktiv bestrålning får man chipet att räkna fel. Största gemensamma delaren mellan det rätta resultatet och det felaktiga blir då *p* eller *q*.

Beskrivningen av angreppssätt kan göras mycket längre. Metoden rönte också motstånd i början. Den svenske matematikern Tore Herlestam påpekade, att det fanns så många metoder som en angripare kunde använda, $p - 1$,

$p + 1$ och många fler. För att försvåra detta måste konstruktören av kryptot ta många hänsyn vid lottningen av p och q. Detta skulle göra nyckelframställning nästan lika svår som forcering.

När metoden med elliptiska kurvor blev känd, blev allt paradoxalt nog enklare. Vilket primtal som helst kunde vara känsligt, bara på grund av tur, utan någon särskild insats från angriparen. Då räcker nu att göra denna lyckliga chans försumbart liten och det går med tillräckligt långa nycklar.

10.2 Genombrott i faktorisering?

Matematiker är duktiga. Någon kanske kommer på något, som alla andra har förbisett. Vad händer, om ett okänt snille hittar en radikalt snabbare metod för faktorisering, kanske med polynominell tid? Många grupper skulle drabbas.

I försvaret blir nyckelhanteringen fel. Allt kan läsas! Fienden kan se all planering. Ett ovisst antal hemligheter skulle vara förrådda för lång tid framöver.

För bankerna och deras kunder blir konsekvensen att bankkonton kan öppnas. Kunderna kan inte lita på att deras konton stämmer. Internettjänster måste stängas omedelbart.

En rad andra applikationer skulle drabbas. De som trodde sig ha säker e-post skulle inte längre ha det.

10.3 Reparera skadan

Vad gör en säkerhetsansvarig som ställts inför ett nytt angrepp?

Först och främst: Han håller tyst så länge som möjligt för att vinna tid. Hemligheten kanske ännu inte har nått alla dem som vill åt pengarna.

På kort sikt måste han kartlägga var metoden faktiskt används. I gynnsammaste fall kunde han fördubbla nyckellängderna. Känsliga tillämpningar måste temporärt stängas av. Autenticering och signering får ersättas av alternativa säkerhetslösningar, om det inte blir nödvändigt att gå tillbaka till helt manuella rutiner. Självklart skulle sådana åtgärder ge en allvarlig PR-förlust och dessutom visa angripare på problemet.

På längre sikt måste den säkerhetsansvarige implementera en ny metod. Han känner inte till alla detaljer, men det finns andra asymmetriska krypteringsmetoder, baserade på t.ex. elliptiska kurvor.

Inom försvarsmakter kan man förvänta sig beredskap mot sådana händelser: Nya algoritmer är utvärderade. Gränssnitt för maskin- och programvara är definierade sedan länge oberoende av krypteringsmetoden. Standarder finns och kan följas. Nya produkter kan med kort varsel tillverkas och installeras.

Sådan beredskap finns inte i det civila samhället. Där måste man i stället vänta på att någon annan väljer ut en ny metod och gör den till standard. Valet är mycket svårt för en vanlig säkerhets- eller datachef.

Sedan måste man anlita någon matematiker för att analysera metodens säkerhet, undersöka prestanda för kryptering och dekryptering, samt hitta metoder för att lotta fram nycklar.

Systemexperter skall sedan fastställa gränssnitt för överföring av nycklar och texter.

Därefter kommer implementering i ett antal miljöer, OS/390, Unix, Windows och aktiva kort. Allt skall testas.

Metoden skall läggas in i alla applikationer och testas där. Nya aktiva kort skall produceras och distribueras.

Så småningom kan systemen produktionssättas och köras parallellt med de gamla. PR-frågor måste hanteras hela tiden. Först efter en tids inkörning kan säkerhetschefen sova lugnt.

En radikalt snabbare faktoriseringsmetod skulle alltså förorsaka en årslång och utomordentligt dyr serie av åtgärder.

10.4 Vad skulle Du göra?

Tänk om just Du hade kommit på något revolutionerande. Vad skulle Du göra? Det finns många alternativ:

1. **Skryta** för vänner och bekanta, men annars göra ingenting
2. **Gå till chefen** och be om en löneförhöjning på 500:-
3. **Lägga ut** nyheten **på nätet** och få igång en diskussion
4. **Skriva en artikel** och skicka den till någon fin tidskrift
5. Läsa in sig på området och **doktorera** (om det inte redan är klart)
6. Som god svensk medborgare **kontakta försvaret**
7. Kontakta **säkerhetschefen på SEB**. Ta betalt för att hålla tyst och hjälpa till att reparera skadan.
8. I samma syfte välja en **stor amerikansk bank** i stället
9. Gå till **maffian** och få procent på stulna konton
10. Kontakta en **utländsk ambassad** som vill skada västs försvarssystem.

Men vilket alternativ man än väljer, så återstår många frågor. Hur bär Du Dig åt för att övertyga? Hur håller Du saken hemlig? Hur sparar Du Dina bevis på ett säkert sätt?

10.5 Råd till ett snille

Historien om Andrew Wiles är lärorik: Hans barndomsdröm var att bevisa Fermats stora sats. Han skaffade sig kunskaperna för det. När han såg en chans att lyckas arbetade han ensam i sju år, berättade ingenting för någon, publicerade ingenting, utom forskningsresultat från andra områden, som han hade på lager och långsamt släppte. Sitt resultat presenterade han på en svit av föreläsningar utan att i förväg ha uttalat sitt syfte. Ändå gällde saken bara den vetenskapliga äran.

Primtalsforskningen har däremot konsekvenser för oss alla. Nya resultat kan störa samhällets ordning. En framgångsrik forskare möter samma etiska problem, om än inte lika allvarliga, som fysikerna kring den första atombomben.

Upptäckaren av en revolutionerande faktoriseringsmetod måste därför vara ännu mer försiktig än Andrew Wiles. Han får inte genom tanklöshet hjälpa ekonomiska brottslingar eller militära fiender.

Lättsinnigt skryt (alternativ 1 – 3) är därför uteslutet, liksom publicering via vanliga kanaler (4, 5). Arbetet, som säkert kommer att ta flera år, måste bedrivas under sträng sekretess. Program och dokumentation kan stjälas och måste gömmas undan på säkert ställe. Kanske inte ens bankfack räcker. Gamla kopior måste förstöras. Datorn måste vara krypterad och sakna varje förbindelse med Internet.

När presentationen förhoppningsvis inför någon institution som skyddar allmänna intressen (alternativ 6 – 8), så måste trovärdigheten garanteras. Helst borde forskaren inte lämna ifrån sig någonting, utan bara begära ett stort sammansatt tal av sin motpart. Han skulle erbjuda sig att avslöja primfaktorerna dagen efter. Han skulle gå hem till sin dator, utföra beräkningarna, lämna ifrån sig faktorerna och direkt efteråt förstöra hårddisken, så att den inte kan bli stulen.

En säkerhetschef måste vara mycket kunnig för att gå med på detta. Troligen väntar han sig i stället en rapport, som han kan skicka till någon expert inom universitetsvärld eller försvar. Forskaren bör därför ha sin rapport färdigskriven redan inför den första presentationen.

När motparten är övertygad om att idén är riktig och åtgärder nödvändiga, måste forskaren tänka över ett framtida avtal. Förutom de ekonomiska villkoren måste hans prioritet till upptäckten garanteras. Han skall ha rätt att i sinom tid uttala sig för världspressen och skriva memoarer. Han fysiska säkerhet måste hela tiden skyddas. För detta kan han lova tystnad och expertkonsultationer om alternativa metoder under något år.

Skulle han däremot välja de mindre hedrande alternativen 9 eller 10, så måste han dessutom besinna faran för sitt eget liv. Han måste hota med att metoden kommer ut om han skulle bli mördad. Endast fantasin sätter gränsen för vad som skulle behövas av spionromansliknande hysch-hysch.

10.6 En anekdot

För ett antal år sedan skrev jag ett faktoriseringspaket. Huvudmetoderna var Pollard ρ-metoden och det kvadratiska sållet. Det senare klarade att faktorisera svåra tal om 54 siffror på en natt. Jag ville mera. Jag tvekade mellan vilka förbättringar som jag borde implementera och vilka inköp jag kunde göra. Därför analyserade jag metoden mer detaljerat än förut. Formlerna pekade på ett genombrott!

Tankar från förra avsnittet dök upp. Var det sant? Skulle jag programmera direkt eller planera i detalj? Vem skulle jag informera? Vilka försiktighetsmått skulle jag vidta?

Jag skrev och skrev, 40 sidor på engelska. Jag trodde mig kunna klara knappt dubbelt så många siffror som tidigare. Ingen katastrof för samhället var i sikte, men kanske ett världsrekord för mig. Det skulle räcka med 10 PC under helg för att slå den tidens rekord, uppnått med hjälp av talkroppssållet.

Det var svårt att hålla tyst. Till slut visade jag det för professor Johan Håstad på NADA. Han såg felet på någon timme. Det roliga var slut.

Litet optimism måste man ha när man forskar. Man måste tro på sig själv en tid om man skall komma framåt. Misstag är förlåtliga för ämnet är så roligt. Kanske Gauss önskan i det inledande citatet går i uppfyllelse helt och fullt och någon annan hittar en metod som är både snabb och sann.

11 Referenser

11.1 Baskunskaper

Birkhoff – Mac Lane, A survey of modern Algebra, Macmillan, New York, 1960.
Gammal lärobok jag hade hemma, finns nog modernare

Hardy – Wright, An Introduction to the Theory of Numbers, Fifth Edition, Oxford Science Publications, 2000.
Bred klassiker inom talteorin

LeVeque, Topics in Number Theory Vol. I, Addison – Wesley, Reading, Massachusetts, 1958.
Gammal lärobok jag hade hemma, finns nog modernare

11.2 Översiktsböcker

Crandall and Pomerance, Prime Numbers, a Computational Perspective, Springer Verlag, New York 2001.
Modern och pedagogisk sammanfattning av hela området. Ganska översiktlig, bör kompletteras med grundläggande fakta och originalartiklar med bevis.

Knuth, Seminumerical Algorithms, Addison-Wesley, Reading, Massachusetts, Första upplagan 1971, Andra upplagan 1981.
”Bibeln” inom algoritmer. Jämför första och upplagan, och se vad forskningen gått framåt!

Hans Riesel, En bok om primtal, Studentlitteratur 1968.
Kort pionjärverk med beräkningsaspekten. Nu kan vem som helst med modern programvara uppnå samma resultat.

Hans Riesel, Prime Numbers and Computer Methods for Factorization, Birkhäuser, Boston, Basel, Stuttgart 1987.
Stor utvidgning jämfört med förra boken. Mycket baskunskaper ges i ett i appendix. Stor kärlek till ämnet. Boken saknar elliptiska kurvor och talkroppssåll.

11.3 Specialområden

A. K. Lenstra, H. W. Lenstra, Jr: Algorithms in Number Theory, Kap. 12 i Handbook of Theoretical Computer Science, Elsevier Sience Publishers 1990.
Mycket koncentrerad artikel som täcker nästan allt.

A. K. Lenstra, H. W. Lenstra, Jr: (Eds.) The development of the number field sieve, Springer Verlag, Berlin och Heidelberg 1993.
130 sidor med originalartiklar om talkroppssållet

B. Riemann, Über die Anzahl der Primzahlen unter einer gegebenen Grösse, Monatsberichte der Berliner Akademie, November 1859, transcribed by D. R. Wilkins, preliminary version.
http://www.emis.de/classics/Riemann/Zeta.pdf

11.4 Opublicerade artiklar på Internet

http://www.cse.iitk.ac.in/news/primality.pdf
Agrawal, Kayal, Saxena: Primes in P, Department of Computer Science & Engineering, Indian Institute of Technology Kanpur, INDIA, 2002.
Deterministiskt, ovillkorligt primtalsbevis i polynominell tid.

A. G. L. Atkin and D.J. Bernstein: Prime sieves using binary quadratic forms
http://cr.yp.to/papers/primesieves.pdf
I stället för Eratostenes' såll!

D. J. Bernstein: Circuits for integer factorisation: A proposal
http://cr.yp.to/papers/nfscircuit.pdf
Specialhårdvara är lönsam!

D. J. Bernstein: Arbitrarily tight bounds on the distribution of smooth integers
http://cr.yp.to/papers/psi.pdf
Uppskattningar av antalet glatta tal

A. Shamir, E. Tromer: Factoring Large Numbers with the TWIRL Device (draft)
http://www.wisdom.weizmann.ac.il/~tromer/papers/twirl.pdf
Hur man kan bygga en faktoriseringsmaskin

11.5 Rekord och kuriosa

http://www.utm.edu/research/primes
Underbar site med rekord och bevis!

http://www.crypto-world.com/FactorAnnouncements.html
Länkar till många rekord

http://www.maths.ex.ac.uk/~mwatkins/zeta/
Många fakta om ζ-funktionen med figurer och skisser till bevis. Dessutom länkar till tvivelaktiga bevis för Riemanns förmodan.

ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Richard.Brent/champs.txt
Rekord för elliptiska kurvor

http://www.ercim.org/publication/Ercim_News/enw49/franke.html,
Rekordet för talkroppssållet

http://www.mersenne.org/prime.htm
Sökning efter Mersennetal

http://www.zetagrid.net/servlet/service/statistic
Beräkning av nollställen till ζ-funktionen

http://www.top500.org/lists/2002/11/
De f.n. största datorerna

http://www.pbs.org/wgbh/nova/proof/wiles.html
En banbrytande matematiker berättar om sin upptäckt

http://www.trnicely.net/index.html
Om primtalsgap och Pentiumbuggen