CSE291-14: The Number Field Sieve

https://cseweb.ucsd.edu/classes/wi22/cse291-14

Emmanuel Thomé

January 6, 2022

Part 1c

Old factoring algorithms

Factoring with simple sieving

The product tree approach

Recap from last time

- We know how to test primality.
- We learned how to find primes in an interval with sieving.
- When we do so, the size of the sieve array matters.
- We may use a segmented version of the sieve of Eratosthenes in order to alleviate the memory concerns. With segmentation, random access is possible: we can search for primes in a range $[a, b]$ without enumerating all primes up to a. Enumeration up to $\sqrt{b}$ is enough.

Plan

Factoring with simple sieving

The product tree approach

More information with sieving

Instead of storing a zero in the array, one can keep further information.

Depending on the information stored, one can get more or less data on the factorization of the integers, at a cost of higher memory:

- Initialize with zero, and add one at each sieving step. Gives the number of distinct prime factors. Requires $B \log_2 \log_2 B$ bits of memory.
- With only 2 bits per position, one can get the numbers that contain exactly two distinct primes.

Sieving for factoring integers

Can we recover the full factorization of integers with sieving?

The exact goal needs to be stated.

Factorization with elementary sieving

Assume all primes below B are known.
In a sieve region $\mathcal{R} = [A, 2A]$ (for example):

- Goal 1: for each $N \in \mathcal{R}$, find its prime factors below B.
- Goal 2: find all $N \in \mathcal{R}$ whose prime factors are all below B. These are called smooth numbers.
- Goal 3: like goal 2, but list the prime factors of smooth numbers, too.

Whether our goal is 1, 2, or 3, we have an array with a value tied to each $n \in \mathcal{R}$.

Finding smooth parts with sieving

To achieve goal 1:

- Initialize the array cell indexed by n with the integer n itself.
- Whenever this index is identified as a multiple of a prime below B, divide it (perhaps several times), and store the information about the divided values.
- Eventually, information at index n gives the prime divisors of n below B, as well as the cofactor

This is expensive.

- Memory: $O(\#\mathcal{R} \cdot \log_2 n)$ (which is also the size of the output).
- Arithmetic cost is large as well, with many divisions.

Identifying smooth integers with sieving

To achieve goal 2:

- Initialize the array cell indexed by n with an approximation of $\log n$.
- When sieving, subtract $\log p$ at the sieved position.
- In the end, positions with a small remaining value are likely to be smooth.
- Some caveats: rounding, prime powers.

This is a very simple, yet very important mechanism.

Cost: $O(\#\mathcal{R})$ approximated values, and $O(\#\mathcal{R} \cdot \log\log B)$ additions/subtractions.

Improvements discussed at length, especially when $\mathcal{R}$ is large.

Factoring smooth integers with sieving

To achieve goal 3, one option is to do as in goal 1, and filter the results.

Better:

- Do detection first (as in goal 2).
- In a second step, do re-sieving, but keep information only for the indices that we know are smooth.

This is very worthwhile when smooth numbers are rare.

This re-sieving technique will appear (much) later on in the NFS context as well.

Plan

Factoring with simple sieving

The product tree approach

Batch smoothness detection

Fact. The Sieve of Eratosthenes relies on two properties:

- The set of numbers to test for primality (or for smoothness) has a structure: *arithmetic progression*.
- The set of primes we consider has a structure: *all the primes up to a bound*.

What can we do with less or no structure? Such situations exist:

- Coppersmith's variant of NFS with several number fields: *tested numbers have no structure*
- Large-prime separation: *primes in an interval, and some prime are forbidden*

Shopping list

Main tool. Asymptotically fast integer arithmetic. Using FFT-based techniques, integers of n bits can be multiplied, divided in almost linear time. Same for GCD.

Notation. $\mathrm{M}(n)$ is the cost of multiplying two integers of n bits. Division costs $O(\mathrm{M}(n))$ and GCD costs $O(\mathrm{M}(n) \log n)$.

Note: asymptotically fast multiplication algorithms are readily available in software. The GNU multiprecision library (GMP), for example, has an implementation of the Schönhage-Strassen.

The multiplication of two integers of one billion bits each takes about. . . (your guess).

Batch'ed trial division

Input.

- A set of integers $x_1, \ldots, x_k$;
- A factor base $\mathcal{F}$, i.e. a set of primes $p_1, \ldots, p_\ell$;

Output. The x_i that are $\mathcal{F}$-smooth.

Idea. Compute the GCD of x_i with $\prod p_j$.

If these GCD are computed sequentially, we get a quadratic complexity.

Rationale. Try to do operations between integers of the same size.

First step: $\prod p_j$

The approach is pretty bold.

Multiply all p_j's together!
Multiply all x_i's together!
Collect winnings.

$P = \prod p_j$ a big number. The product of all primes below 2^B is a $2^{B+0.53}$-bit integer.

Fortunately, fast integer arithmetic is not only useful in theory, it is also useful in practice!

Strategies to compute $P = \Pi p_j$

Naive. Even with fast multiplication, $\prod p_j$ costs a quadratic bit-complexity.

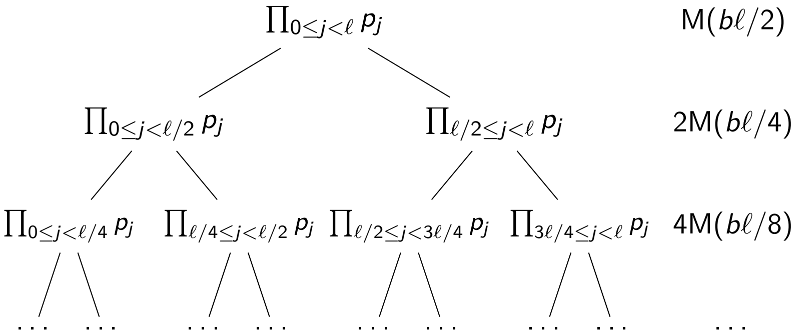
Subproduct-tree.

Assume the number ℓ of p_j is a power of 2, and build a binary tree, from leaves that are the primes. Do a multiplication at each node.

- Same number of multiplications;
- All of them are balanced (the two operands have the same size).

Complexity. If all primes have b bits, total cost is $O(\mathrm{M}(\ell b) \log \ell)$.

First step: $\prod p_j$

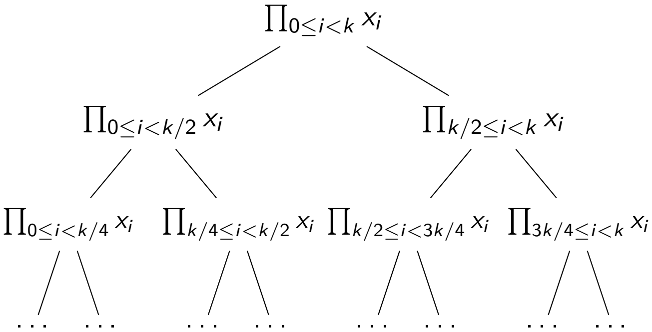


Although quasi-linear, multiplication is still supra-linear:

$4\mathrm{M}(b\ell/8) \le 2\mathrm{M}(b\ell/4) \le \mathrm{M}(b\ell/2)$.

Consequence. If M is close to linear, same cost at each level.

Do the same for x_i

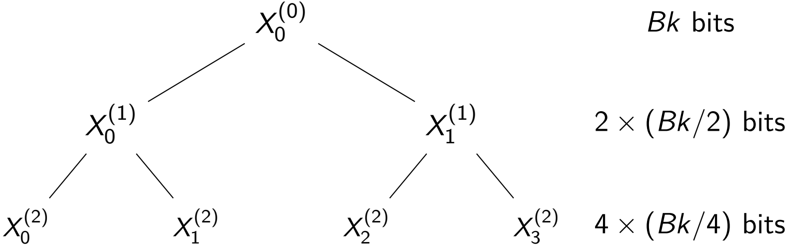


Rem. If the x_i have B bits, cost of this construction is $O(\mathsf{M}(Bk)\log k)$.

Important. Keep the whole tree in memory (and lose a log factor in space complexity).

The tree of the x_i stays in memory

Let $X_i^{(r)}$ denotes the i-th node at depth r from the root.



We have a single large integer on top, and a large collection of small integers at the bottom.

Descend the remainder tree

Let $P = \prod p_j$ that has been computed.

Remember that we want the GCD of P with the leaves x_i.

Idea. Compute $P \bmod x_i$ before taking these GCD. For that, we descend P along the remainder tree of the x_i's.

Let $R_i^{(r)} = P \bmod X_i^{(r)}$. Since $X_i^{(r-1)} = X_{2i}^{(r)} X_{2i+1}^{(r)}$, we have

$$R_{2i}^{(r)} = R_i^{(r-1)} \bmod X_{2i}^{(r)} \quad \text{and} \quad R_{2i+1}^{(r)} = R_i^{(r-1)} \bmod X_{2i+1}^{(r)}.$$

$$R_0^{(0)} = P \bmod X_0^{(0)}$$

$$R_0^{(1)} = R_0^{(0)} \bmod X_0^{(1)} \qquad R_0^{(1)} = R_1^{(0)} \bmod X_1^{(1)}$$

$$R_0^{(2)} \qquad R_1^{(2)} \qquad R_2^{(2)} \qquad R_3^{(2)}$$

Descend the remainder tree

At level r, this will cost 2^r divisions of an integer of size $Bk/2^{r-1}$ and one of size $Bk/2^r$. Close to $O(\mathsf{M}(Bk))$.

Again, the total cost is $O(\mathsf{M}(Bk)\log k)$ plus $O(\mathsf{M}(Bk, b\ell))$ for the first step.

Conclusion. We can compute all the GCD of $\prod p_j$ and the x_i in a time that is quasi-linear in the input size.

Is it practical? YES!!! Several fun stories related to that. (try to search "GCD all the keys" or something similar).

Left as exercise. Handle powers, deduce the full factorization.
Everything follows more or less easily, in the same complexity.
Main reference: D. J. Bernstein.
Sage scripts avalaible at `https://facthacks.cr.yp.to/`

Wrap up

- Prime testing is not a difficulty for the usage range that we target.
- Sieving is unsurprisingly a very basic building block that will resurface. Remember resieving.
- Batch smoothness detection is a very interesting tool. It's not only fun, but also useful in NFS context.

CSE291-14: The Number Field Sieve

https://cseweb.ucsd.edu/classes/wi22/cse291-14

Emmanuel Thomé

January 6, 2022

Part 1d

Old factoring algorithms

Pollard ρ

$p - 1$ and $p + 1$

ECM

Mundane factoring needs within NFS

Enumerating primes gives a method to factor a number.
This is called trial division (TD).

TD works only to some extent

The required time to trial-divide N by all prime numbers below B with sieving is roughly $\widetilde{O}(B \log N)$

Many other integer factorization algorithms can be used, with the common characteristic:

- Runtime is polynomial in $\log N$.
- $\frac{\text{runtime}}{\text{success probability}}$ is (at most) exponential in $\log B$.

Within NFS, these algorithms are used to obtain auxiliary factorization of many intermediate numbers.

- Pollard rho.
- $p-1$, $p+1$, the Elliptic Curve Method (ECM);

Second goal: combinations of congruences

A second class of algorithms is given by those whose complexity depends only on N (albeit super-polynomially).

These algorithms are the precursors of the lineage that culminates with NFS.

- Fermat factoring.
- Dixon random squares method.
- CFRAC: the continued fraction method.
- QS: the quadratic sieve.

Plan

Pollard ρ

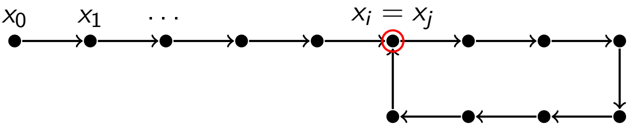
$p - 1$ and $p + 1$

ECM

Pollard ρ

Idea. Pick k random elements x_i modulo N. If two of them are equal modulo p, then $\gcd(x_i - x_j, N)$ is likely to give p.

Trick. The number of GCD to test is quadratic. To avoid that, use a pseudo-random sequence $x_{i+1} = f(x_i)$ and cycle detection.



Analysis. Birthday paradox says that the first collision occurs after $O(\sqrt{p})$ elements. More rigourous: use properties of the functional graph of f. AofA can find constant in $O()$.

Pollard ρ

Implementation.

- Use $f(x) = x^2 + c$. This gives enough randomness (quantified with expander graph theory).
- Use Floyd cycle detection: run two sequences in parallel, one going twice as fast as the other. Lose at most a factor of 2.

The overall complexity is $O(\sqrt{p})$ operation modulo N to extract a factor p.

In the worst case, where N is RSA, this gives $O(N^{1/4})$.

Pollard-Strassen

Pollard ρ is heuristic.

In a nearby (asymptotic) complexity ballpark, the Pollard-Strassen is slightly more expensive, but proven and not heuristic.

Pollard-Strassen

We are after a prime factor $p < B$ of N. Let $C = \left\lceil \sqrt{B} \right\rceil$.

- Compute $P = \prod_{i=0}^{C-1} (X - i) \in \mathbb{Z}/N\mathbb{Z}[X]$. Keep product tree.

  This costs $\mathrm{M}(C) \log C$ operations.

- Compute $Q = \prod_{i=0}^{C-1} (X + Ci) = (-C)^C P(-X/C)$.

  This costs $O(C)$ operations.

- Multi-evaluate Q at the roots of P.

  This costs $\mathrm{M}(C) \log C$ operations.

- Any evaluation that has a non-trivial gcd with N narrows down a potential factor to a range of size C.

PS is not useful in practice, but is a fun application of asymptotically fast algorithms!

Plan

Pollard ρ

$p - 1$ and $p + 1$

ECM

$p-1$ and $p+1$

Another family of algorithms.

- One aspect in common with ρ.

  "Something" happens $\mod p \longrightarrow$ detect it $\mod N$.
- The gist of it is how "something" is defined.
- Some algebra is involved.

Pollard $p - 1$ method

Idea: assume $p \mid N$ and a is prime to p. Then

$$(p \mid a^{p-1} - 1 \text{ and } p \mid N) \Rightarrow p \mid \gcd(a^{p-1} - 1, N).$$

Same if some R is known s.t. $p - 1 \mid R$ and we compute

$$\gcd((a^R \bmod N) - 1, N).$$

How do we find R ? Only reasonable hope is that $p - 1 \mid B!$ for some (small) B. In other words, $p - 1$ is B-smooth.

Algorithm: $R = \prod_{p^\alpha \leq B} p^\alpha = \text{lcm}(2, \ldots, B)$.

Our "**something**" is the event $a^R \equiv 1 \mod p$.

$p - 1$ is one-shot

$p - 1$ succeeds if N is divisible by some p with $p - 1$ smooth.

If, for a given N, $p - 1$ failed to find a factor, you need to find another algorithm to factor it.

If we fix B and consider many integers N_i with one known factor $\approx 2^x$, $p - 1$ will return a factor for a fixed fraction of the input† and will fail for the rest.

† $\frac{\psi(2^x, B)}{2^x}$ (see lecture about smoothness)

The $p+1$ method (Williams, Guy, ...)

Idea. Work in an extension of degree 2 of $\mathbb{F}_p$.

The multiplicative group is of order $p^2 - 1 = (p-1)(p+1)$, and the subgroup $\mathbb{T}_2(p)$ of elements of norm 1 is of order $p+1$.

Difficulties.

- We do not know p; can not be sure to work in a genuine field extension.
- How do we work with elements of norm 1 anyway?

$p+1$: working around difficulties

Implicit representation of $T_2(p)$

Fact: if θ is a root of $x^2 - Ax + 1 \mod p$, then:

- the other root is $1/\theta$.
- if $D = A^2 - 4$ is a non-square, θ is an element of norm 1 in $\mathbb{F}_p(\sqrt{D}) \approx \mathbb{F}_{p^2}$ (i.e., an element of $\mathbb{T}_2(p)$).
- if D is a square, θ is simply an element of $\mathbb{F}_p$. We'll be redoing $p-1$ inadvertently.

In effect, we're choosing (the shape of) θ first, and D afterwards.

- We don't know if $\sqrt{D}$ defines $\mathbb{F}_{p^2}$ or not.
- In either case, if $p - \left(\frac{D}{p}\right)$ is B-smooth, then $\theta^{B!} \equiv 1 \mod p$.
- How do we compute with θ^n?

$p + 1$: computing θ^n

We only need to care about $v_n = \theta^n + \theta^{-n} \in \mathbb{F}_p$. We have:

$$v_0 = 2, \quad v_1 = A, \quad v_{m+n} = v_m v_n - v_{m-n}.$$

We use a Montgomery ladder to compute $\{v_n, v_{n+1}\}$.

$\{v_n, v_{n+1}\} \to \{v_{2n}, v_{2n+1}\}$.	$\{v_n, v_{n+1}\} \to \{v_{2n+1}, v_{2n+2}\}$.
$v_{2n} = v_n^2 - v_0,$	$v_{2n+1} = v_n v_{n+1} - v_1,$
$v_{2n+1} = v_n v_{n+1} - v_1.$	$v_{2n+2} = v_{n+1}^2 - v_0.$

Very similar to standard binary powering

Requires $\log_2 n$ mult and $\log_2 n$ squares modulo N. (compared to $4.5 \log_2 n$ operations with naive representation of $\mathbb{F}_p(\sqrt{D})$).

$p + 1$: summary

As the $p - 1$ method, this is a one-shot method.

It is not a lot more expensive than the $p - 1$ method.

However, it brings something new only 50% of the time.

Can this be generalized:

- Yes, but generalizations with field extensions don't work as well.
  - See Factoring with cyclotomic polynomials.
  - Work in algebraic tori which are varieties of dimension $\phi(d)$.
- The "good" generalization is ECM. Curves are varieties of dimension 1.

Handling close misses

The $p - 1$ and $p + 1$ algorithms succeed if
$p \pm 1 = \text{small} \times \text{small} \times \cdots \times \text{small}$.

What happens if $p \pm 1 = \text{small} \times \text{small} \times \cdots \times \text{medium}$?

Assuming there is only one extra factor of medium size, it can be caught with the Stage 2 algorithms.

Second phase: the classical one

Let $b = a^R \bmod N$ and $\gcd(b, N) = 1$.

Hyp. $p - 1 = Qs$ with $Q \mid R$ and s prime, $B_1 < s \leq B_2$.

Test: is $\gcd(b^s - 1, N) > 1$ for some s.

Let s_j denote the j-th prime. In practice all $s_{j+1} - s_j$ are small (Cramer's conjecture: $s_{j+1} - s_j \leq (\log B_2)^2$).

- Precompute $c_\delta \equiv b^\delta \bmod N$ for all possible δ (small);
- Compute next value with one multiplication:

$$b^{s_{j+1}} = b^{s_j} c_{s_{j+1} - s_j} \bmod N.$$

Cost: $O((\log B_2)^2) + O(\log s_1) + (\pi(B_2) - \pi(B_1))$ multiplications $+(\pi(B_2) - \pi(B_1))$ gcd's. When $B_2 \gg B_1$, $\pi(B_2)$ dominates.

Rem. We need to enumerate all primes $< B_2$; use a table of size $O(B_2)$ or a low-memory (segmented) Eratosthenes sieve.

Second phase: faster

Select $w \approx \sqrt{B_2}$, $v_1 = \lceil B_1/w \rceil$, $v_2 = \lceil B_2/w \rceil$.

Write our prime s as $s = vw - u$, with $0 \leq u < w$, $v_1 \leq v \leq v_2$.
One has $\gcd(b^s - 1, N) > 1$ iff $\gcd(b^{wv} - b^u, N) > 1$.

1. Precompute $b^u \bmod N$ for all $0 \leq u < w$.
2. Precompute all $(b^w)^v$ for all $v_1 \leq v \leq v_2$.
3. For all u and all v evaluate $\gcd(b^{vw} - b^u, N)$.

Number of multiplications is
$w + (v_2 - v_1) + O(\log_2 w) = O(\sqrt{B_2})$, memory is also $O(\sqrt{B_2})$.

Number of *gcd* is still $\pi(B_2) - \pi(B_1)$.

Second phase: faster

Algorithm:
- Compute $h(X) = \prod_{0 \le u < w}(X - b^u) \in \mathbb{Z}/N\mathbb{Z}[X]$
- Evaluate all $h((b^w)^v)$ for all $v_1 \le v \le v_2$.
- Evaluate all $\gcd(h(b^{wv}), N)$.

Analysis, using product trees:

Step 1: $O((\log w)\mathrm{M}(w))$ operations (product tree).

Step 2: $O((\log w)\mathrm{M}(\log N))$ for b^w; $v_2 - v_1$ for $(b^w)^v$; multi-point evaluation on w points takes $O((\log w)\mathrm{M}(w))$.

Rem. Evaluating $h(X)$ along a geometric progression of length w takes $O(w \log w)$ operations (see Montgomery-Silverman).

Total cost: $O((\log w)\mathrm{M}(w)) = O(B_2^{0.5+o(1)})$.

Plan

Pollard ρ

$p-1$ and $p+1$

ECM

The ECM algorithm

The starting observation is that Pollard $p-1$ is nice, but of limited use.

- Pollard $p-1$ implicitly uses $\mathbb{F}_p^*$. And there is only one $\mathbb{F}_p^*$ per p.
- The $p+1$ algorithm uses another group which therefore increases the factoring chances.
- But it basically stops here.

What ECM achieves is that it works works with a structure defined modulo p (and therefore computable implicitly with arithmetic modulo N), which has many different instances.

This is done with Elliptic curves.

The ECM algorithm

ECM: the Elliptic Curve factoring Method. ECM = a variant of $p \pm 1$ which is Not a one-shot algorithm.

An elliptic curve: set of solutions of certain algebraic systems.

- $y^2 = x^3 + ax + b$ (with constants a, b).
- $By^2 = x^3 + Ax^2 + x$ (with constants A, B).
- $x^2 + y^2 = 1 + dx^2y^2$ (with constant d).
- $x^2 + y^2 = 1$ and $ax^2 + z^2 = 1$ (with constant a).

All are different ways to define isomorphic mathematical objects.

Elliptic curves over $\mathbb{F}_p$

Given an equation that defines an elliptic curve E:
$E(\mathbb{F}_p) = \{\text{points with coordinates in } \mathbb{F}_p\} \cup \{\infty\}$.

- Elements of $E(\mathbb{F}_p)$ can be represented easily.
- $E(\mathbb{F}_p)$ forms a finite group with easily computable group law.
- $p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}$.
- If we work modulo N, we also work in $E(\mathbb{F}_p)$ implicitly.
- A match in $E(\mathbb{F}_p)$ can be detected with arithmetic modulo N by gcd.

The common notation is to write the group law on elliptic curves additively.

- ∞ is the neutral element.
- $P + Q$ is the group law.
- $[n]P$ is done by double-and-add (a.k.a. square-and-multiply, additively).

ECM: adapting $p - 1$

ECM works in the same way as $p - 1$ and $p + 1$.

- Pick a curve and a point P on it.
  Important: do it in one go, e.g.

$$x, y, a \in_R \mathbb{Z}/N\mathbb{Z}, \text{ then let } b = y^2 - x^3 - ax.$$

- Hope for $\#E(\mathbb{F}_p)$ to be B_1-smooth for a divisor p of N.
- Compute $[B_1!]P$. Test (gcd) if something happened mod p.
- Stage 2 works, too.
- If no factor found, start over with a new curve.

Arithmetic

ECM is de facto the home of most improvements on these $(p-1)$-like factoring methods.

- Arithmetic: a lot of time is spent in computations modulo N. It is worthwhile to use trade-offs (such as Montgomery multiplication), and size-specific code.
- Some ways to choose an elliptic curves and a point on them are better than others (we can force some small factors in $\#E(\mathbb{F}_p)$).
- For fixed B_1, there are ways to compute $[B_1!]P$ slightly more efficiently than by square-and-multiply (Lucas chains, PRAC).

ECM performance

ECM finds a factor $\approx p$ of an n-bit integer N in time:

$$\exp\left(\sqrt{2}(\log p)^{1/2}(\log\log p)^{1/2}(1+o(1))\right) \times \mathrm{M}(n).$$

This is called a sub-exponential complexity (w.r.t $\log p$).

ECM is very efficient for $p \approx 10^{30\ldots40}$, record of $p \approx 10^{83}$.

ECM can be distributed massively.

Reference implementation: GMP-ECM (Paul Zimmermann).

Note: we'll see this funny kind of complexity again!

ECM: fun and profit

Factoring enthusiasts like big ECM hits.

Yearly top ten table here.

It's the beginning of January, the best time of the year to enter that table (at least temporarily).

- A few core-years and/or luck to make it to the table (60 digits).
- A few dozen core-years to find a 70 digit factor.