Factoring large integers using Quadratic Sieve

Christopher Teljstedt Henrik Viksten

chte@kth.se hviksten@kth.se

November 11, 2013

Abstract

This report will give you an insight to factorization of integers using quadratic sieve. It covers the basics of the factoring problem and mentions other commonly used algorithms, but the main focus is on the quadratic sieve which is the best algorithm for this problem up to 110 bit integers.
This report was written during the course DD2440 Advanced Algorithms at KTH 2013 and recieved the maximum amount of points in the scoring system KATTIS.

Contents

I. Introduction

This report will cover the work of a integer factorization program written in the course DD2440 advanced algorithms. It was decided to focus primarily on quadratic sieve which should give a good understand of the problem in general.

I.1 Purpose

The goal of the program is to be able to factorize integers in an efficient way and also pass KATTIS test cases with as high score as possible, this is done by improving the program step-by-step and gradually increasing the performance and intelligence in the task at hand. In the report the methods used will be analyzed described. How they work independently and their correlation in our implementation.

I.2 Problem

An unknown set of 100 integers in varying size are used as input to the algorithm from KATTIS. The output is either the factors of the input if it is solved or the string fail otherwise. If number $n_i = p_1^{k_1} \dots p_m^{k_m}$, then the program should print the following on `stdout`.

$$\left.\begin{array}{l} p_1\\ p_1\\ \vdots\\ p_1\end{array}\right\} \quad k_1 \text{ times}$$

$$\vdots$$

$$\left.\begin{array}{l} p_m\\ p_m\\ \vdots\\ p_m\end{array}\right\} \quad k_m \text{ times}$$

One of the problems is dealing with large integers within the timeframe but also finding every single non-trivial factor. The restrictions in kattis are 15 seconds and 64MB of memory.

I.3 Scope

There is no intention of reinventing any methods that already exists, the idea is to create a solver that can factorize unknown integers and in the end get a high score on KATTIS.

I.4 Statement of Collaboration

Some text.

II. Preliminaries

Notation. By '$\log_b$' we denote the base b logarithm and the natural logarithm denotes by ln=$\log_e$ with $e \approx 2.71828$. The largest integer $\leq x$ is denoted by '[x]'. The number of primes $\leq x$ is denoted by '$\pi(x)$', and due the *Prime number theorem*[1] we know that $\pi(x) \approx x/\ln(x)$.

Smoothness. A positive integer is *B-smooth* if all its prime factors are lesser than a boundary B. Furthermore an integer, say x, is said to be *smooth with respect to S*, if x can be completely factored using integers by some set S alone.

Modular arithmetic. Throughout this paper '$x \equiv y \bmod n$' means that $(x-y)$ is a multiple of n whereas x, y and $n \in \mathbb{N}_{\neq 0}$.

Similarly, '$x \not\equiv y \bmod n$' mean that $(x-y)$ is not a multiple of n. Euclid's algorithm for finding the *greatest common divisor* of two non-negative integers, say x and z, is denoted by $\gcd(x,z)$.

III. Background

The Integer Factorization problem is defined as follows; given a composite integer n, find any non-trivial factor e of n, such that $e|n$. At first it might seem like a trivial task, but for large integers this has proven to be a very difficult task.

III.1 Trial Division

The most straight-forward approach to factoring composite numbers is *trail division*, which essentially (just like its name suggests) just check for every prime number $p \leq \sqrt{n}$ if $p|n$.

Improvement is to only do tests when p is a prime number, i.e. keep a list of precomputed prime numbers. Although trial division is easy to implement and guaranteed to grant an answer it is not efficient with large integers. There is improvements that can be applied such as storing a precomputed tables of primes to bring up the speed. However, this instead requires alot of memory during run-time as well as storage, when dealing with larger integers. A

III.2 Pollard's ρ

In 1975 John M. Pollard proposed a new and very efficient Monte Carlo algorithm for factoring integers, now known as Pollard's ρ (rho) method. It was a break through and proved to be alot more faster than its predecessor trial division for finding small non-trivial factors of a large integer n.

III.2.1 Basic idea

Pollard's p basic concept is that a sequence of pseudo-random integers constructed as

$$x_0 = \texttt{rand}(0,\ n-1) \tag{1}$$

$$x_i = f(x_{i-1}) \ (\texttt{mod}\ n),\ \texttt{for}\ i = 1, 2, \ldots \tag{2}$$

where f is polynomial which in most practical implementations has the form $f(x) = x^2 + \alpha, \alpha \neq 0, -2$. The key observation that if we consider a non-trivial divisor of n, say d, that is small compared to n. Then there exists smaller congruence groups modulo d compared to n. Because of this there is a probability that there exists two x_i and x_j such that $x_i \equiv x_j \pmod{d}$, while $x_i \not\equiv x_j \pmod{n}$. Thus it follows from that $\gcd(x_i - x_j, n)$ is a non-trivial factor of n.

The idea of Pollard ρ algorithm is to iterate this formula until it falls into a cycle. We want to find a x and y whereas the x makes twice as many iterations as the y using the $f(x) \pmod{n}$ as a generator of a pseudo-random sequence. The $gcd(x - y, n)$ is taken each step, and it reaches n we have not found an answer and the algorithm terminates with a failure.

We can write $n = p \cdot q$, when x, which iterates twice as fast as y, catches up with y which will happen eventually, at this point factor p will be found. The time it will take cannot be proven matematically and can only be proven by heuristics. If the sequence behaves randomly it would take approximately p steps to find p, which is not very effcient. [2]

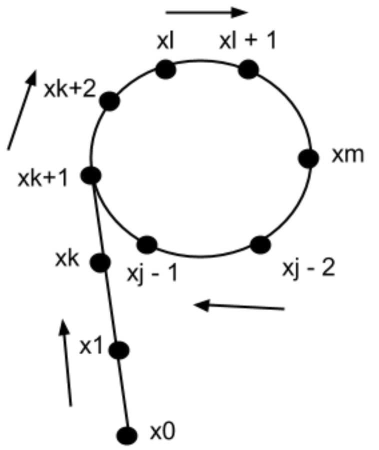

Illustration of the Pollard's Rho cycle. The mapping of x_{i+1} is instead replaced with a function $x^2 + 1$ so that we get $x_i^2 + 1$ (mod n) the factor p will be found after $O(\sqrt{p})$ ϵ $O(n^{1/4})$ steps. [2]

Pollard ρ algorithm

```
pollards $\rho(n, X_0)$
$a \leftarrow X_0$
$b \leftarrow X_0$
**while** $factor = 1$ **do**
  $a \leftarrow f(a)$
  $b \leftarrow f(f(b))$
  $factor \leftarrow gcd(|a - b|, n)$
**end while**
**if** $factor = n$ **then**
  {print 'fail'}
  **return** $-1$
**else if** $factor > 1$ **then**
  **return** $factor$
**end if**
```

Pollard's rho algorithm can be improved further by implementing Brent's cycle finding method. [5]

III.2.2 Optimizating with Brent

Brent's algorithm The idea behind the optimization is to only perform one function call to $f(x)$ in every iteration, instead of three. This can be achieved by using Brent's method to find cycle instead of Floyd's.

Floyd's algorithm to find cycles lets b make twice as many iterations than a, i.e. moving twice as fast. Instead Brent's algorithm is trying to find the smallest power of two that is larger than the length of the cycle.

```
pollards $\rho$ $(n, X_0)$
$power \leftarrow 1$
$iterations \leftarrow 1$
**while** $factor = 1$ **do**
  **for** $i = 0$ to $iterationerpergcd$ **do**
    **if** $power = iterations$ **then**
      $a \leftarrow b$
      $power \leftarrow power * 2$
      $iterations \leftarrow 0$
    **end if**
    $b \leftarrow f(b)$
    $iterations \leftarrow iterations + 1$
    $factor \leftarrow factor * |a - b|$
  **end for**
  $factor = gcd(factor, n)$
**end while**
**if** $factor = n$ **then**
  {try new start value}
  **return** pollards rho$(n, X_0 + 1)$
**else if** $factor > 1$ **then**
  **return** $factor$
**end if**
```

III.3 Miller-Rabin primality test

Miller-Rabin primality test is a probabalistics algorithm which determines if the given input n is a prime. It is based on the properties of strong pseudoprimes, given an integer n, $n = 2^r * s + 1$ where s is odd, you choose a random number a with the properties $1 \leq a \leq n - 1$. When $a^s \equiv 1$ (mod n) or $a^{2js} \equiv -1$ (mod n) where $0 \leq j \leq r - 1$, if the input number n is a prime it will pass the test with any random number a.

Since Miller-Rabin is a probabalistics method it is not completely true that n is a prime simply by passing the test, however the probability that the answer is true when n is a composite number is $1/4^n$ which grows quickly with n. It can be considered a very small trade-off to use a probabalistics method because the algorithm executes at $O(k\ log^3 n)$ where k is the number of different values of a tested. [6]

IV. Quadratic Sieve

IV.1 Brief explaination

The quadratic sieve algorithm is today the algorithm of choice when factoring very large composite numbers with no small factors. The general idea behind quadratic factoring is based on Fermat's observation that a composite number n can be factored if one can find two find two integers $x, y \in Z$, such that $x^2 \equiv x^2 \pmod{n}$ and $x \not\equiv \pm y \pmod{n}$. This would imply that,

$$n \mid x^2 - y^2 = (x-y) \cdot (x+y) \tag{3}$$

but n neither divides $(x-y)$ nor $(x+y)$. Furthermore it can be rewritten as $(x-y) \cdot (x+y) = k \cdot p \cdot q$ for some integer k, thus becoming two possible cases.

- either p divides $(x-y)$ and q divides $(x+y)$, or vice versa.
- or both p and q divides $(x-y)$ and neither of them divides $(x+y)$, or vice versa.

Hence, the greatest common divisor of $(x-y, n)$ and $(x+y, n)$ would in first case yield p or q and a non-trivial factor of n is found. In the second case we get n or 1 and trivial solution is found. There is atleast $1/2$ probabilty of the solution being non-trivial[3].

Carl Pomerance suggested a method to find such squares[3]. The first step in doing so is to is to define the polynomial

$$q(r) = (r + \lfloor\sqrt{n}\rfloor)^2 - n \approx \tilde{r}^2 - n \tag{4}$$

Now, consider we set of primes $P = \{p_1, p_2, ..., p_k\}$ lesser than a bound B, i.e $k < \pi(B)$. We then want to construct a subset of integers $r_1, r_2, \ldots, r_k$ such that $\forall i : q(r_i) = r_i^2 - n \pmod{n}$ is smooth in respect to P, more specifically $\forall i$:

$$q(r_i) = r_i^2 \equiv p_1^{e_{i1}} \cdot \ldots \cdot p_k^{e_{i\pi(B)}} \pmod{n} \tag{5}$$

where e_{ij} is the exponent of p_j of factorization of $q(r_i)$. If the exponents for all primes sums to an even number we arrove at following relation:

$$\prod_{i=1}^{n} q(r_i) = \prod_{i=1}^{n} r_i^2 \equiv (p_1^{e_1} \cdot \ldots \cdot p_k^{\pi(B)})^2 \pmod{n} \tag{6}$$

and the integers we x, y we sought to find are simply:

$$x = \prod_{i=1}^{n} q(r_i) \tag{7}$$

$$y = \prod_{i=1}^{\pi(B)} p_1^{e_i} \tag{8}$$

Lets consider a factor base P with the primes $p_1, p_2, ..., p_k$ that is $k \geq B$ and co-prime to n. We want to search for small r_i so that $q(r_i)$ is smooth in respect to P. If we find such r's we say that $q(r_i) = (r + \lfloor\sqrt{n}\rfloor)^2 - n$ is B-smooth and we can factor completely over the factor base.

A prime factorization $p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_{\pi(B)}}$ of a B-smooth number can then be expressed be entirely expressed by the factor base and exponent vector $(e_1, e_2, \ldots, e_{\pi(B)})$.

Since a prime p only divides $q(r_i)$ if and only if it divides $q(r_i + kp)$ for any integer k, we can found these values efficiently using a sieve. For this reason its called the *sievingstep* and because only primes $(\frac{n}{p_i}) = 1$ can divide $q(r_i)$, explains the definition of the factor base P. Naivly one could simply randomly select r in a range of interest and verify that $q(r)$ is divisable by all the primes in the factor base. Instead in quadratic sieve algorithm we can first solve the quadratic congruence $r^2 \equiv n$ (mod p) and then cleverly only divide by prime numbers corresponding to the interval r and $r + kp$ for some any integer k. Thus, finding these r's becomes easier than just a simple trail division as we apriori know at that a B-smooth $q(r_i)$ is divisable by a subset of primes in P. This is the reason why trail division of B-smooth numbers preferable over is a plain naive trail division.

Then the product of subsequence of x i.e. $q(x_1) \cdot q(x_2) \cdot \ldots \cdot q(x_k)$ produced a square iff the exponent vectors has only even entries. That is that it's the *null*-vector in mod 2 space. So with the collection of smooth numbers we want to form their exponent vectors, reduce them modulo 2. For that vector space (finite field of two elements) the sequence of vectors thus becomes linearly dependent and can be solved with Gaussian Eliminatio to find the non-empty subsequences. The exponent vectors is represented as the rows of a matrix. Hence, the column sums must be even.

IV.2 Deciding on factor base

To factorize during the sieving step we want all the $q(r)$'s to be smooth in respect to the factore base, i.e. $q(r)$ must be divisable by only primes in the factor base. If a prime p divides $q(r_i)$ earlier mentioned polynomial implies following

$$(r_i + \lfloor \sqrt{n} \rfloor)^2 \equiv n \texttt{ mod } p_i \tag{9}$$

Hence, n is a quadratic residue modulo p_i and we only need to consider those primes. Thus the set of primes p_i for which the Legendre symbol $(\frac{n}{p_i})$ is 1 will form the factor base[3].

IV.3 Sieving step

Only a small fraction of numbers will be completely factorized by the primes of the factor base. Therefore, a crucial step is to sieve as many numbers as possible. From following observation we can decide on intervals of primes in P that to divide prime numbers of $q(r)$.

$$\begin{aligned} q(r) &= \tilde{r}^2 - n, \ \tilde{r} = (\lfloor \sqrt{n} \rfloor + r) && (10) \\ q(r+kp) &= (\tilde{r} + kp)^2 - n && (11) \\ q(r+kp) &= \tilde{r}^2 + 2\,\tilde{r}kp + (kp)^2 - n && (12) \\ q(r+kp) &= q(r) + 2\tilde{r}kp + (kp)^2 && (13) \\ &\equiv q(r) \pmod{p} && (14) \end{aligned}$$

So the solution $q(r) \equiv 0p$ for r_i yields a sequence $q(r_i)$ that is divisable by p. This can be solved using Shanks-Tonelli algorithm[3]. Thus we obtain two solutions which we call idx_{1p} and $idx_{2p} = p - idx_{1p}$. Then those $q(r_i)$ with r_i's in the sieving interval are divisable by p when $r_i = idx_{1p}, idx_{2p} + pk$ for some integer k.

Gauss-eliminatio If $q(r_i)$ does completely factor, the next step is to test different linear combination of these so that it yields perfect square for product of $q(r_i), i \in [1, k]$. In other words we want to find solutions to

$$q(r_1)e_1 + q(r_2)e_2 + \ldots + q(r_k)e_k \qquad (15)$$

where e_i is either 0 or 1. This means that if $\vec{a}_i$ is the row in a matrix A corresponding to $q(r_i)$ then we get

$$\vec{a}_1 e_1 + \vec{a}_2 e_2 + \ldots + \vec{a}_k e_k \qquad (16)$$

and we have to solve

$$\vec{e}_1 A^T = \vec{0} \; (\texttt{mod} N) \qquad (17)$$

where $\vec{e}$ is the exponent vector. Then the task simply becomes to find the nullspace of A^T, with Gauss Elimination in the field $GF[2]$ so that all calculations can be made modulo 2.

To ensure linear dependancy there have to be more B-smooth numbers r than the number of primes in the factor base. We want to produce linear combination so that each prime power in the combination is even, only then we have a perfect square.

From this stage what remains is to try combinations until we have a solution of linear combinations yielding a product that is perfect square, with equation (7) and (8). To ensure that x and y are not trivial solutions we can make use of equation (1) and check the requirement $x \not\equiv \pm y$ (mod n), since there is otherwise a $1/2$ probability of $\gcd(x - y, n)$ resulting in a trivial solution. If a trivial solution is found we just rerun and try another linear combination until we find a non-trivial one.

V. Implementation

For our implementation we used several papers as reference. One that really come handy lecture notes by Carl Pomerance [4]. We implemented a naive quadratic sieve using only single polynomial paradigm.

The program can be divided into three general phases:

- calculating smoothness bound and generate factor base
- sieving
- gaussian elimination

VI. Results

These tests were solely based on the score and running time on the `KATTIS` problem `oldkattis:factoring`.

Primality testing At first we just tried to do a primality test with Miller-Rabin. The results can be seen in table **??**

Quadratic sieve We implemented a naive Quadratic Sieve from start and at first with bad B smooth boundary. It had unoptmizied sieving functions, we did handle perfect powers.

This yielded a result on Kattis as shown below. **??**

By improving on the problems mentioned above and tweaking the smoothness boundary B which regulates the factor base size aswell as the number of linear relations, i.e how many extra smooth numbers, we managed to get 100 points score on Kattis.

Primality testing At first we just tried to do a primality test with Miller-Rabin. The results can be seen in table **??**

VI.1 Kattis submission

The best Kattis submission has the ID:459013.

VII. Discussion

We chose to work with quadratic sieve since it's one of the most efficient integer factorization algorithms out there. It is the fastest algorithm for factorization up to approximately 110 bit integers, there was no doubt it would be a challenge, but since we had knowledge that Pollard's Rho combined with Brent's cycle finding granted approximately 75 points in KATTIS and that the code was not a great challenge to write we wanted to give ourselves a challenge and write a state of the art algorithm. We gave ourselves the time to investigate the more complex alternative and ended up succeeding but still have the basic knowledge about Pollard's Rho since that was our lifeline if our implementation would not have worked.

The result was as expected from a correctly implemented quadratic sieve method, however, on the first submit (that was working) we got a score of 85 points which was due to partly a lack of general optimization but the main reason was making sure the smoothness bounds worked correctly. There are a lot of mathematical parts to implementing this algorithm, altough not very complex math in theory getting all the parts of the algorithm to work well was a tough job. The understanding of the concept came as we went along and step-by-step we improved our code and realized the earlier errors we made and in the end realized that the concept is not hard to graps if you divide the problem in to subproblems and follow pre-existing formulas to execute it.

VIII. Conclusion

In hindsight there is no regret of choosing the more complex solution to the problem, it did not only give us more understanding of how to solve the problem we were given in an intelligent way but we also managed to successfully implement the quadratic sieve algorithm to achieve maximum score in KATTIS. Of course the solution can be optimized further but considering the time constraints we are satisfied with our work.

References

[1] G.H. Hardy, E.M. Wright, D.R. Heath-Brown, and J. Silverman. *An Introduction to the Theory of Numbers*. Oxford mathematics. OUP Oxford, 2008.

[2] Johan Hastad. Notes for the course advanced algorithms, 2000.

[3] Carl Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer Berlin Heidelberg, 1985.

[4] Carl Pomerance. Smooth numbers and the quadratic sieve. In *in [Buhler and Stevenhagen 2007]. Citations in this document: §5*. University Press, 2005.

[5] Weissteinm Eric W. Brent's factorization method.

[6] Weissteinm Eric W. Miller-rabin strong pseudoprime test.