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 100 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. 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

As described in the scope the intention is not to find a new method of factoring integers but instead use existing methods and implement them in a intelligent way.

III.1 Trial Division

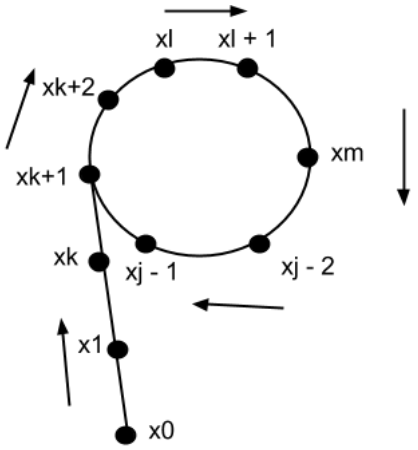
Trial division is an easy implemented algorithm for finding factors to N. Like its name suggests it uses tries to divide N by using a $x > 2$. If the remainder of $N/x = 0$ the factor is a non-trivial factor, if we cannot find a divisor x that gives the remainder 0 then N is a prime.

The restrictions for the algorithm are easy to find and improve, $N > 1$ and $x > 2$. This can be improved by saying that $N > x$ and since the factors of N cannot be larger than $\sqrt{N}$ we can also say that $\sqrt{N} > x > 2$. Another improvement is to only do tests when x is a prime number. Although trial division is easy to implement and guaranteed to grant an answer it is not efficient for large a N.

III.2 Pollard's Rho

Pollard rho is a prime factorization algorithm which is based on Floyd's cycle-finding algorithm. The idea of pollard rho algorithm is to iterate a formula until it falls into a cycle. We want to find a x and y where the x makes twice as many iterations as the y using a function $moduloN$ as a generator of a pseudo-random sequence. The $GCD|x - y|$ of N is taken each step, and the GCD reaches N we have not found an answer and the algorithm terminates with a failure.

We can write $N = p * q$, 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]



The figure above illustrates 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's rho algorithm can be improved further by implementing Brent's cycle finding method. [4]

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,

$$n \mid x^2 - y^2 = (x - y) \cdot (x + y) \qquad (1)$$

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.

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

$$q(x) = (x + \lfloor \sqrt{n} \rfloor)^2 - n \tag{2}$$

One might realize that this satisfies the relation $(x + \lfloor \sqrt{n} \rfloor)^2 \equiv q(x) \pmod{n}$.

Now, consider we have integers $x_1, x_2, \ldots, x_k$ for which so that the product $q(x_1) \cdot q(x_1) \cdot \ldots \cdot q(x_n)$ is a perfect square, and call it y^2. Then if we let $x = (x_1 + \lfloor \sqrt{n} \rfloor) \cdot (x_2 + \lfloor \sqrt{n} \rfloor) \ldots (x_k + \lfloor \sqrt{n} \rfloor)$ we have a solution $x^2 \equiv y^2$ that satisfies equation (1).

To find such x's we must realize that we if a prime p divides $q(x_i)$ then $(x_1 + \lfloor \sqrt{n} \rfloor)^2 \equiv n \pmod{p}$. Hence, n is a quadratic residue modulo p 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. This set of primes is a tool for factoring which we will further on referr as *factor base*.

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

A prime factorization $p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_{\pi(B)}}$ of a B-smooth number v can then be expressed as, we can express as a exponent vector $e(v) = (a_1, a_2, \ldots, a_{\pi(B)})$ and we arrive at following formula.

$$y^2 \equiv \prod_{i=1}^{\pi(B)} p_i^{e_i v} \tag{3}$$

Then the product of subsequence of v 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. The linear combinations producing a perfect square can the be solved with Gaussian Eliminations modulo 2 if the exponent vectors is represented as the rows of a matrix. Thus, the column sums must be even.

IV.2 Brief explaination

V. Implementation

VI. Results

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

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 Håstad. 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] Weissteinm Eric W. Brent's factorization method.