Int. J. Contemp. Math. Sciences, Vol. 6, 2011, no. 2, 59 - 64

Generalized Trial Division

Murat ŞAHİN

Ankara University
Faculty of Sciences
Department of Mathematics
06100, Tandoğan, Ankara, Turkey
muratsahin1907@gmail.com

Abstract

In this paper, we propose an algorithm of factoring any integer N. This special-purpose algorithm will factor a number N efficiently if there exist a positive integer k such that k times any prime factor of N is close to $\sqrt{N}$. it will be improved by using nested squares polynomials. Indeed, under an assumption about nested squares polynomials, this algorithm becomes an general-purpose algorithm with sub-exponential or polynomial running time.

Mathematics Subject Classification: 11Y05, 11Y16

Keywords: integer factorization; special purpose algorithm; nested squares polynomials

1 Introduction

The fundamental theorem of arithmetic states that every positive integer can be written uniquely as a product of primes, when the primes in the product are written in nondecreasing order. Integer factorization is a classical problem in number theory. Although remarkable progress have been achieved, especially in the last thirty years, this problem is still considered difficult.

The security of several cryptographic systems is based on the hardness of integer factorization problem. Among them, the RSA system [5] is most famous.

So far, the fastest general-purpose factorization algorithm is the number field sieve [1]. There are also many different special-purpose factoring algorithms, including Pollard's $p-1$ method, $p+1$ method, Fermat's method, special number field sieve [2].

Special-purpose factorization algorithms are very efficient at factoring numbers of special form. In section 2 will be discussed trial division and fermat factorization which are special-purpose factoring algorithms.

In section 3 will be proposed a special-purpose factoring algorithm which is called generalized trial division and it will be compared with fermat factorization and trial division algorithms. Generalized trial division will be improved by an nested square polynomial in section 4.

2 Some Special-Purpose Factoring Algorithms

General-purpose factoring algorithms are the running time depends mainly on the size of N, the number to be factored, and is not strongly dependent on the size of the factor p found. In contrast, some factoring algorithms are tailored to perform better when the integer N being factored is of special form; these are called special-purpose factoring algorithms. The running times of such algorithms typically depend on certain properties of factors of N. Examples of special -purpose factoring algorithms include trial division and fermat factorization algorithms. Here we summarize these two factoring algorithms.

2.1 Trial Division

The simplest factorization algorithm is trial division algorithm. Trial division will always have its place in factoring, regardless of advances in algorithms.

Trial division consists of dividing N by every odd integer up to $\left\lfloor\sqrt{N}\right\rfloor$. if N has small prime divisor p then we can find p easily by trial division algorithm.

In trial division, one can search the prime factor p dividing N by every odd integer from $\left\lfloor\sqrt{N}\right\rfloor$ down to 2. In this case, if p is close to $\sqrt{N}$ then trial division efficiently find the prime factor p. So, trial division considered special-purpose factoring algorithm.

Algorithm 1 (Trial Division)
Input : N
Output: Factors of N
1. for s from 2 to $\left\lfloor\sqrt{N}\right\rfloor$
2. if s divides N then
3. return s, N/s
4. end if
5. end for

2.2 Fermat Factorization

An old strategy to factor a number is to express it as the difference of two nonconsecutive squares. if one can write N in the form $a^2 - b^2$, where a,b are nonnegative integers, then one can immediately factor N as $(a+b)(a-b)$. if $a - b > 1$ then the factorization is nontrivial.

To formalize this as an algorithm, we take $a = \left\lceil \sqrt{N} \right\rceil, \left\lceil \sqrt{N} \right\rceil + 1, \left\lceil \sqrt{N} \right\rceil + 2, ...$ values and check whether $a^2 - N$ is a square. if it is, say b^2, then we have $N = a^2 - b^2 = (a+b)(a-b)$. For odd numbers N that are product of two nearby integers or have a factor of nearby $\sqrt{N}$, it is easy to factor N by Fermat factorization.

There are various tricks that can be used to speed up the fermat factorization algorithm. For example, by congruences it may be differentiated that some residue classes for a make it possible for $a^2 - N$ to be a square. Eg, if $N \equiv 1 \pmod 4$, then a can not be even, or if $N \equiv 2 \pmod 3$, then a must be multiple of 3.

Also, a multiplier might be used. As we know, if N is product of two nearby integers, then Fermat factorization method finds this factorization quickly. Even if N does not have this property, it may be possible for kn to be a product of two nearby integers, and $\gcd(kN, N)$ may be taken to obtain the factorization of N. For example, take $N = 2581$. Fermat factorization method starts with $a = 51$ and does not terminate until the ninth choice, $a = 59$, where we find that

$$59^2 - 2581 = 900 = 30^2$$

and

$$2581 = 89.29$$

. But, if we try Fermat factorization method on $3N = 7743$, we terminate on the first choice, $a = 88$, giving $b = 1$. Thus,

$$3N = 89.37$$

and note that

$$89 = \gcd(89, N) \ , \ 29 = \gcd(87, N).$$

(For a general discussion, see page 191-192 in [3]).

The running time of the Fermat factorization algorithm is exponential, but basic idea of almost all known sub-exponential algorithms, such as, quadratic sieve, Dixon, continued fraction and Number field sieve algorithms etc. depend on Fermat factorization method.

Algorithm 2 (Fermat Factorization)
Input : N
Output: factors of N
1. for a from $\left\lceil \sqrt{N} \right\rceil$ to N

2. $bsquare = a * a - N$
3. if isSquare($bsquare$) then
4. $b = \sqrt{bsquare}$
5. $s = a - b$
6. $t = a + b$
7. if $s \neq 1$ and $s \neq N$ then
8. return s, t
9. end if
10. end if
11.end for

Here the isSquare(z) function is true if z is a square number and false otherwise. It is straightforward to construct an isSquare function by taking a square root, rounding the answer to an integer, squaring the result, and checking if the original number is reproduced.

3 Generalized Trial Division

Our algorithm can be considered generalization of trial division. Suppose we want to factor N, we take trial values of the number a from the sequence

$$\left\lfloor \sqrt{N} \right\rfloor, \left\lfloor \sqrt{N} \right\rfloor \pm 1, \left\lfloor \sqrt{N} \right\rfloor \pm 2, \left\lfloor \sqrt{N} \right\rfloor \pm 3, ...$$

and check whether greatest common divisor of a and N is a proper divisor of N, where $\left\lfloor \sqrt{N} \right\rfloor$ denotes the greatest integer less than or equal to $\sqrt{N}$.

Let $N = pq$ with p, q primes and $p < q$. The algorithm takes minimum of $\left\lfloor \sqrt{N} \right\rfloor - p$ and $q - \left\lfloor \sqrt{N} \right\rfloor$ steps. So, this algorithm works well when p close to $\sqrt{N}$. In the same boat, trial division and fermat factorization are better than this algorithm. In trial division, if we search the prime factor p from $\left\lfloor \sqrt{N} \right\rfloor$ down to 2 then trial division is obviously better than generalized trial division. On the other hand, Fermat factorization requires about

$$\frac{p+q}{2} - \sqrt{N} = \frac{p + \frac{N}{p}}{2} - \sqrt{N} = \frac{\left(\sqrt{N} - p\right)^2}{2p}$$

steps. Hence, fermat factorization is also better than generalized trial division. But, generalized trial division works well not only when p is close to $\sqrt{N}$ but also if there exist a positive integer $k > 1$ such that k times any prime factor of N is close to $\sqrt{N}$.

Generalized trial division works well when there exist a positive integer $k > 0$ such that kp or kq is close to $\sqrt{N}$. Lets see this. Assume without loss of generality that $kp = \sqrt{N} \pm c_0$ where c_0 small. The algorithm finds prime factor p in c_0 steps because of greatest common divisor kp and N is p.

Algorithm 3 (Generalized Trial Division)
Input : N
Output: Factors of N
1. for i from 1 to $\left\lfloor \sqrt{N} \right\rfloor$
2. $a_1 = \left\lfloor \sqrt{N} \right\rfloor + 1$
3. $a_2 = \left\lfloor \sqrt{N} \right\rfloor - 1$
4. $A_1 = \gcd(N, a_1)$
5. $A_2 = \gcd(N, a_2)$
6. if A_j divides N then $(j = 1, 2)$.
7. return A_j, N/A_j
8. end if
9. end for

Example 1 *Factor* $N = pq = 8403414598759$. *For* $a = 2898863 = \left\lfloor \sqrt{N} \right\rfloor - 1$; $\gcd(a, N) = 263533$ *is a prime factor of* N. *Hence, generalized trial division factor this number in two steps. The other prime factor of* N *is* 31887523. *Generalized trial division algorithm is effective for this number because there exist positive integer* k *such that* $k * 263533$ *is very close the* $\sqrt{N}$ *(for this example* $k = 11$*).*

4 Nested Squares Polynomials

The interesting algebraic identity

$$\left(\left(x^2 - 85\right)^2 - 4176\right)^2 - 2880^2 = \left(x^2 - 1^2\right)\left(x^2 - 7^2\right)\left(x^2 - 11^2\right)\left(x^2 - 13^2\right)$$

discovered by R. E. Crandall, allows the evaluations of a product of 8 integers by a succession of 3 squares and 3 substractions [4]. There are infinitely many such identities which are called nested squares of length 3. There exist also identities which are nested squares of length 4, for example,

$$f = \left(\left(\left(x^2 - A\right)^2 - B\right)^2 - C\right)^2 - D^2 = \prod_{i=1}^{8}\left(x^2 - a_i^2\right),$$

Where $a_1 = 139$, $a_2 = 317$, $a_3 = 541$, $a_4 = 719$, $a_5 = 827$, $a_6 = 953$, $a_7 = 1049$, $a_8 = 1087$, $A = 600445$, $B = 172337340816$, $C = 16685576724080968704000$ and $D = 10660920170019569049600$. This identity allows finding 16 algebraic factors with only 4 squarings.

We can improve generalized trial division using such a f polynomial. In generalized trial division, for a number $N = pq$ to be factored one could simply

take $\gcd(f(x), N)$ where the number x from the sequence

$$\left\lfloor \sqrt{N} \right\rfloor, \left\lfloor \sqrt{N} \right\rfloor \pm 1, \left\lfloor \sqrt{N} \right\rfloor \pm 2, \ldots .$$

Hence, we improved the algorithm about a multiple of 32.

5 Conclusion

We propose an algorithm of factoring any integer N. This special-purpose algorithm will factor a number N efficiently if there exist a positive integer k such that k times any prime factor of N is close to $\sqrt{N}$. It will be improved by using nested squares polynomials. The importance of this algorithm is that if one can find the nested squares of length k such that k is sufficiently large , then the algorithm becomes general-purpose algorithm with sub-exponential or polynomial running time according to k.

References

[1] A. Lenstra and H. Lenstra Jr, editors. The development of the number field sieve, Lecture Notes in Mathematics. Springer-Verlag, 1554, 1993.

[2] Peter L. Montgomery. A Survey of Modern Integer Factorization Algorithms, CWI Quarterly, 1994, 7(4), 337-365.

[3] R. Crandall and C. pomerance, Prime numbers: a computational perspective, Springer-Verlag, New York, 2001

[4] R. E. Crandall, Topics in advanced scientific computation, TELOS, The Electronic Library of Science, Springer, New York, 1996.

[5] R. Rivest, A. Shamir and L. Adleman. A method for obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, 1978, 21, 120-126.

Received: August, 2010