A DETRMINISTIC FACTORIZATION AND PRIMALITY TESTING ALGORITHM FOR INTEGERS OF THE FORM Z MOD 6 = -1

Noureldien A. Noureldien[1], Mahmud Awadelkariem[1], DeiaEldien M. Ahmed[1]

[1]Department of Computer Science and Information Technology, University of Science and Technology, Omdurman, Sudan

{Noureldien@hotmail.com, mah_awad@gmail.com, Diamahamed@gmail.com}

Abstract

Prime numbers are known to be in one of two series; P mod 6 = ±1. In this paper, we introduce the concept of Integer Absolute Position in prime series, and we use the concept to develop a structure for composite integer numbers in the prime series P mod 6 = -1.

We use the developed structure to state theorems and to develop a deterministic algorithm that can test simultaneously for primality and prime factors of integers of the form Z mod 6 = -1.

The developed algorithm is compared with some of the well known factorization algorithms. The results show that the developed algorithm performs well when the two factors are close to each other.

Although the current version of the algorithm is of complexity ($(N/6^2)$ ½ /2), but the facts that, the algorithm has a parallel characteristics and its performance is dependent on a matrix search algorithm, makes the algorithm competent for achieving better performance.

Keywords: Factorization, Primality Testing; Prime Series; Absolute Position, Relative Position

1. Introduction

Factoring numbers is a hard task, which means that any algorithm designed to factor will not run in polynomial time. In fact, most of the algorithms that exist today run on the order of e^n, where e is Euler's number [1]. Generally speaking, the most useful factoring algorithms fall into one of the following two main classes [2]:

A. The run time depends mainly on the size of N, the number being factored, and is not strongly dependent on the size of the factor found. Examples are: Lehman's algorithm [8] which has a rigorous worst-case run time bound $O(N^{1/3})$, . Shanks's SQUFOF algorithm [9], which has expected run time $O(N^{1/4})$. Shanks's Class Group algorithm [3,4] which has run time $O(N^{1/5+€})$ on the assumption of the Generalised Riemann Hypothesis.

The Continued Fraction algorithm [5] and the Multiple Polynomial Quadratic Sieve algorithm [6], which under plausible assumptions have expected run time $O(\exp(c(\log N \log\log N)^{1/2}))$, where c is a constant (depending on details of the algorithm).

B. The run time depends mainly on the size of f, the factor found. (We can assume that $f \le N$.) Examples are;– The trial division algorithm, which has run time $O(f \cdot (\log N)^2)$. The Pollard "rho" algorithm [7] which under plausible assumptions has expected run time $O(f^{1/2} \cdot (\log N)^2)$. Lenstra's "Elliptic Curve Method" (ECM) [10, 11] which under plausible assumptions has expected run time $O(\exp(c(\log f \log\log f)^{½} . \log(N)^2)$, where c is a constant.

In these examples, the term $(\log N)^2$ is a generous allowance for the cost of performing arithmetic operations on numbers of size O(N) or $O(N^2)$, and could theoretically be replaced by $(\log N)^{1+€}$ for any € > 0.

The fastest known general-purpose factoring algorithm is the General Number Field Sieve (GNFS), which in asymptotic notation takes $S = O(\exp((64/9\ n)^{1/3} (\log n)^{2/3}))$ steps to factor an integer with *n* decimal digits. The running time of the algorithm is bounded below by functions polynomial in *n* and bounded above by functions exponential in *n* [12].

The apparent difficulty of factoring large integers is the basis of some modern cryptographic algorithms. The RSA encryption algorithm [13], and the Blum-Blum Shub cryptographic pseudorandom number generator [14] both rely on the difficulty of factoring large integers. If it were possible to factor products of large prime numbers quickly, these algorithms would be insecure. The SSL encryption used for TCP/IP connections over the World Wide Web relies on the security of the RSA algorithm [15]. Hence if one could factor large integers quickly, "secured" Internet sites would no longer be secure. Finally, in computational complexity theory, it is unknown whether factoring is in the complexity class P. In technical terms, this means that there is no known algorithm for answering the question "Does integer *N*

have a factor less than integer *s*?" in a number of steps that is less than O(*P(n)*) , where *n* is the number of digits in *N*, and *P*(*n*) is a polynomial function. Moreover, no one has proved that such an algorithm exists, or does not exist.

In this paper we present a new approach for developing a primality testing and factorization algorithms. The run time of this algorithm is mainly based on the distant between the two factors, more precisely on the distant between the square root of the integer and the smallest factor.

Although we only deal in this paper with integers of the form Z mod 6 =-1, but the same approach can be applied for integers of the form Z mod 6 = 1. Our proposed approach handles primality testing and prime factorization as one problem, and is based on looking for a prime factor for a given integer within a determined search space. If a factor is found within this space then the given integer is composite otherwise it is prime.

To define the searching space, we state the concept of absolute position for composite numbers of the form Z mod 6=-1, and we use this concept to define an infinite matrix space of absolute positions for composite numbers. In this structure or matrix space, a composite number is represented by its absolute position, and its location in the matrix space is defined by its prime factors. To determine the matrix space searching boundaries for a given integer we state and proof some theorems based on absolute position concept.

Based on the stated theory we present a deterministic primality testing and factorization algorithm by constructing a simple equation that correlates the absolute position of the integer under testing and its prime factors.

This paper contributes to number theory efforts in introducing new concepts and approaches to develop algorithms that improve the complexity of primality testing and factorization algorithms.

This paper is organized as follows; in section two we define prime series and state the basic compositeness theorem for composite numbers of the form Z mod 6 =-1. In section three we develop and highlight a structure of those composites using the concept of absolute position. In section four we discuss the developed absolute positions structure. Based on this structure a preliminary algorithm for Primality testing and factorization was presented in section five. Conclusions are given in section six.

2. Prime Series

It is a well known fact that prime numbers falls either in R5={z: z mod 6 = 5, z € Z} or R1={z : z mod 6 =1, z € Z}. We call R5 and R1 prime series, where R5 and R1 represent integers of the for Z mod 6 =-1 and Z mod 6 =1 respectively. Thus R5 = {5, 11, 17, 23, 29, 35, 41, 47, .} and R1 = {1, 7, 13, 19, 25, 31, 37, 43, 49, ……}. To understand the nature of the composite numbers in R5, we state the following theorem.

Theorem (1)

For any integer Z € R5, then either Z is a prime or composite number. If Z is a composite number, then it either has the form Z= mn, where m € R5 and n € R1. Or Z has more than two factors, in which case, either Z has the form Z = m_1m_2…..m_i, where m_1,m_2,…..,m_i € R5 and i =3,5,7,,9 .., or it has the form Z= m_1m_2…..$m_i.n_1n_2$……n_j , where m_1,m_2,…..,m_i € R5 i=3,5,7,9,.., and n_1,n_2,……,n_j € R1 j=1,2,3,4,5….

Proof:

We only need to prove the theorem for compositeness.

(a) Suppose Z € R5 is a composite number that has two factors, then

$$Z=mn \rightarrow mn \pmod 6 = 5 \rightarrow (((m \bmod 6)(n \bmod 6)) \bmod 6) = 5 \quad (1)$$

Since (m mod 6) and (n mod 6) is either 0,1,2,3,4 or 5,. (1) holds only if either (m mod 6) =5 and (n mod 6) =1 or (m mod 6) =1 and (n mod 6) =5. Therefore one factor (m or n) € R5 and the other factor € R1.

(b) If Z is a composite number with more than two factors, then Z can either be expressed as:

$$Z= m_1m_2\ldots..m_i, \text{ where } m_1, m2,\ldots.., mi \in R5 \quad (2)$$

Or

$$Z = m_1m_2\ldots..m_i.n_1n_2\ldots\ldots n_j, \text{ where } m_1,m_2,\ldots..,m_i \in R5 \text{ and } n_1,n_2,\ldots\ldots,n_j \in R1. \quad (3)$$

Clearly (2) holds only when i=3,5,7,9,…, and (3) holds only when i=3,5,7,9,… and j=1,2,3,4,5,6,….

3. The Structure of Composite Numbers in R5

Based on theorem (1), each prime number in R5 generates a chain of composite numbers in R5 in arithmetic progression. The following Lemma defines chains generated by primes in R5.

Lemma (1)

If P is a prime in R5, then P generates the following chain of composites in R5:

$$Zp= \{ z = 6nP+P; n=1,2,3,4,\ldots\} \quad (4)$$

Thus any composite number Z generated by P is given by:

$$Z = P(6n+1), \text{ where P is the prime generator, } n=1,2,3, \quad (5)$$

Table (1) shows part of the composite numbers matrix space generated by primes in R5.

TABLE (1) Matrix Space for Composite Numbers in R5

R5	5	11	17	23	29	35	41	…	R1
	35	77	119	161	203	245	287	…	7
	65	143	221	299	377	455	533	…	13
	95	209	323	437	551	665	779	…	19
	125	275	425	575	725	875	1025	…	25
	155	341	527	713	899	1085	1271	…	31
	:	:	:	:	:	:	:	:	:

Definition (1): We call the position of each number within the R5 series as an Absolute Position (AP) and for a given integer Z this position is calculated by the following equation:

$$AP(Z) = Z \text{ div } 6 \quad (6)$$

Thus AP (5) = 0.

Definition (2): We call the position of a composite number Z within its prime chain as a Relative Position (RP), and for a given number Z this position is calculated from (5) as:

$$AP(Z) = ((Z \text{ div } P) - 1) \text{ div } 6 \quad (7)$$

Obviously, all primes have RP=0. For example, to find the 15^{th} composite number in chain 17, use (5) to calculate

$Z = p(6n + 1) = 17(6.15 + 1) = 17.91 = 1547$

Theorem (2):

The relationship between the Absolute Position (AP) and the Relative Position (RP) for a composite number Z generated by the prime P is given by:

$$AP(Z) = P * RP(Z) + (P \text{ div } 6) \quad (8)$$

For example, using (8), the AP for the 15^{th} composite number in chain 17 is:

$AP = 15 * 17 + (17 \text{ div } 6) = 255 + 2 = 257$

Using (6), $AP = (1547 \text{ div } 6) = 257$

4. The Structure of Absolute Positions for R5 Composite Numbers

The Absolute Positions for composite numbers in R5 are distributed along an infinite matrix space in a regular manner. Table (2) shows this matrix space.

TABLE (2): Distribution of Composite Number's Absolute Positions in R5

P	11	17	23	29	35	41	47	.	Q	A
AP	1	2	3	4	5	6	7	.		P
	12	19	26	33	40	47	54	.	7	1
	23	36	49	62	75	88	101	.	13	2
	34	53	72	91	110	129	148	.	19	3
	45	70	95	120	145	170	195	.	25	4
	56	87	118	149	180	211	242	.	31	5
	67	104	141	178	215	252	289	.	37	6
	78	121	164	207	250	293	336	.	43	7
		:	:	:	:	:	:	:	:	:

From the structure of Table (2), we note the following:

- The AP of the first composite number generated by prime P is given by : P + AP(P).

- The consecutive difference between adjacent column positions is P.
- The AP of the first composite number generated by prime Q is given by : Q + 5*AP(Q)
- The consecutive difference between adjacent row positions is Q.

The position of any entry in the matrix space can be denoted by X_{rc} , where r denotes the RP(X) row wise and c denotes the RP(X) column wise; For example X_{64} is the entry in row 6, column 4 of the absolute positions matrix, which is 141.

Now, if X is an AP in matrix space which is generated column wise by P and row wise by Q, then X satisfies the following two equations:

$$X = P * AP(Q) + AP(P) \quad (9)$$

$$X = Q * AP(P) + 5\ AP(Q) \quad (10)$$

For example the AP of the 7^{TH} element generated by 17 which is the second element generated by 43 can be calculated from (9) as: X = 17 * 7 + 2 = 119 + 2 = 121, and calculated from (10) as: X = 43 * 2 + 5*7 = 86 + 35 = 121.

Now if we denote AP(P)=c and AP(Q)=r then (9) and (10) becomes respectively:

$$X = r\,(6c+5) + c \quad (11)$$

and

$$X = c(6r+1) + 5r \quad (12)$$

From (11) and (12) we get respectively:

$$(X - c) \bmod (6c + 5) = 0 \quad (13)$$

$$(X - 5r) \bmod (6r+1) = 0 \quad (14)$$

5. A Factorization and Primality Testing Algorithm

Given Z as an R5 integer, we can use (13) and (14) to scan the absolute positions matrix space vertically or/and horizontally, to verify whether AP(Z)=X is within the matrix space or not. To determine the search space limits we have to specify an initial value for c and r to represent the lower limit, and to use the fact that the first composite position generated by P (column wise) and Q (row wise) are P+AP(P)= (6c +5) + c and 5*AP(Q) = 5r respectively, as an upper limit.

Now if Z is composite, then its position in the matrix space is defined by the values of AP(P) =c and AP(Q)=r. And since one of the two factors for Z is more nearer to the initial value of c (the R5 factor) or r (the R1 factor), we need an algorithm to scan the search space vertically and horizontally. The following algorithm initialize c and r to 1, bounds the upper search limit to ((6c+5) + c < n) and scans the matrix space forwards vertically and horizontally simultaneously.

Algorithm:

1. Choose a number, Z *in R5*, you wish to factor
2. Let n = Z div 6
3. Is (n mod 5 = 0)
   - YES: Z is composite, the two factors are; f1= 5 and Z/f1.
   - NO? - continue to next step
4. Let c= *r* = 1
5. Let PrimalityFlag = 1.
6. While ((6c+5) +c) < n)
   a. **Is ((n - c) mod (6*c + 5)) = 0**
   - YES: Z is composite, the two factors are; f1=c * 6 +5 and Z/f1, PrimalityFlag=0, End.
   - NO: **b. Is (n - 5*r) mod (6*r + 1)) = 0**
   - YES: Z is composite, the two factors are; f1=r * 6 +1 and Z/f, PrimalityFlag =0, End.
   - NO: increment c and r
7. Go back to step 6
8. Z is Prime.

The algorithm is a parallel algorithm since the searching space [c=1, ((6c+5) + c)) < n)] can be divided to disjoint intervals as value of n is known.

5.1 Algorithm Performance

There are two major performance degradation problems in the above algorithm. The ***first*** problem is when the column and row variables c and r takes a value that they generates before as a composite absolute position, (for example, when c takes the values 5, 10, 15, 20, ….which are generated previously by c=0, or when c takes the values 12,23,34,45, …which are values generated previously by c=1, and so on, or when r takes the values 10,23, 36, 49, …. which are values generated previously by r=2, and so on), which makes the algorithm repeat a non-significant tests.
The ***second*** problem is the low initial value for c and r when Z is a very large prime number, or a composite number with very large factors.

The first performance problem is due to the fact that each prime in R5 generates a chain of composite numbers in R5. Although it is known that the distance between the prime P and its offspring composite numbers is 6P (for example the composites generated by c=0 are c=5,10,15,…. and the composites generated by c=1 are c=12,23,34, …. …. and so on) generally c=i generates c= i + k (6i+5) , k=1,2,3,4,….,

To make the algorithm recognizing all previous prime values the column generator variable P takes in order to avoid non significant tests, this will make the algorithm very complex in terms of storage and speed. Therefore for simplicity and hence performance wise it is better to omit this problem.
To deal with the second problem of low initial values for c and r, we state the following theorem.

Theorem (3):

Let Z be an R5 large composite number with AP (Z) =X. The suitable initial value for column and row variables c and r (most nearer values to the expected column or row value that generates Z) is c=r= sqrt(X/6).
Proof:
From (11) and (12) we have X= 6rc+5r +c. This implies that X > 6rc. Now if we assume c=r (i.e we will initialize c and r to the same value) then $X > 6\ c^2 \rightarrow X/6 > c^2 \rightarrow c < \text{sqrt}\ (X/6)$.

Based on the above theorem it follows that one of Z factors will be generated by a c value greater than the c initial value while the second factor is generated by a c value less than the initial value.

Since the square root of any composite integer number Z leans towards the smallest factor, it is always more efficient to locate the smallest factor by scanning the absolute positions matrix space backwards rather than forwards. Also since the smallest factor may either be in R5 or R1, for a deterministic reliable algorithm, the scanning should be done simultaneously column and row wise.
Based on theorem (3), we introduce the following algorithm.

Deterministic Primality Testing and Factorization Algorithm

1. Choose a number, Z *in R5,* you wish to factor
2. Let n = Z div 6
3. Is (n mod 5 = 0)
    a. YES: Z is composite, the two factors are; f1= 5 and Z/f1.
    b. NO? - continue to next step
4. Let c= *r* = sqrt(n/6)
5. Let PrimalityFlag = 1.
6. While (c >0)
    a. **Is ((n - c) mod (6*c + 5)) = 0**
        - YES: Z is composite, the two factors are; f1=c * 6 +5 and Z/f1. PrimalityFlag=0, End.
        - NO:
    b. Is **(n - 5*r)mod (6*r + 1)) = 0**
        - YES: Z is composite, the two factors are; f1=r * 6 +1 and Z/f1. PrimalityFlag=0, End.
        - NO: decrement c and r
7. Go back to step 6
8. Z is Prime.

The above algorithm works backwards, which will performs poorly if one of the two factors is much smaller than the other, in such case the same algorithm with initial value of 1 for r and c and forwarding scanning will performs better.

5.1.1 Algorithm Complexity and Testing Results

The worst case of the algorithm is when N is prime, in such case the while loop is executed (($N/6^2$) ½) times. The best case is when Z is composite with factors f1 and f2 are twins, in such case the while loop executed only once O(1). The average case is when Z is composite and f1 and f2 are not twins, is this case the while loop executed in average O (($N/6^2$) ½ /2).

It is obvious that the performance of this algorithms depends on the distant (D) between the absolute positions of the prime generators of the composite factors f1 1nd f2. If we assume that f1 and f2 are generated by P and Q with AP(P)=c and AP(Q)=r respectively and f1 < f2 then D= r –c. The exact no of iterations required to fix f1 is ((($N/6^2$) ½) – c).

5.1.2 Testing

We test our algorithm against the well known factoring algorithms, Fermat, Pollard p-1, Pollard-Roh, and Shanks algorithm. All tests were run on an Intel Pentium 4 3.20 GHz processor with 1GB of RAM. Windows XP Service Pack 2 was the host operating system. Version 1.6.0_07 of the Java Runtime Environment was used. Test programs were executed using a command line invocation of the Java client virtual machine. Java Development Kit version 1.6.0_07 was used for compiling test code. Table (3) shows the testing results. The developed algorithm performs when the factors are too close.

TABLE (3): Comparison Between the Developed Algorithm and Known Algorithms

Length in bits	Number	Factors	Fermat	Shanks	Pollard p-1	Pollard rho	Developed Algorithm
32	2213186951	63709 34739	62	31	0	0	0
40	614278415189	1014131 605719	608	78	16	31	15
48	141053907833849	12746687 11065927	811	203	47	31	16
48	103566076470137	10304911 10050167	47	218	31	31	0
60	807759537987786023	784133621 1030129963	319707	1841	545923	218	26005
64	11002930366353704069	3267000013 3367900313	13790	7361	7004	592	11201
64	15273041663564843243	3990032597 3827798719	31216	9875	3759	718	17955
64	15920357810903658149	3990032597 3990032017	0	0	120385	702	0
95	3157138963392170133340 4835491	167102507056669 188934266696639	Too large	Too large	266979	285059	Too large

6. Conclusion

The developed algorithm utilizes a new approach for developing primality testing and factorization algorithms that gives new sights to the factorization and primality testing problems. The parallel characteristic of the developed algorithm and its dependency on a matrix search algorithm makes it competent for a fast performance on refinements.

The major contributions of the developed theory and algorithm are simplicity and ease of implementation, parallelism, and high speed when integer factors are relatively close.

References

1. Diffie, W. and M. E. Hellman. New directions in cryptography. IEEE Trans. Inform. Theory, 22 (6): 644-654, 1976.
2 R. P. Brent, Some parallel algorithms for integer factorization Proc. Europar'99, Toulouse, Sept. 1999. LNCS **1685**, Springer-Verlag, Berlin, 1-22.
3. R. J. Schoof, "Quadratic fields and factorization", in Studieweek Getaltheorie en Computers (edited by J. van de Lune), Math. Centrum, Amsterdam, 1980, 165-206.
4. D. Shanks, "Class number, a theory of factorization, and genera", Proc. Symp. Pure Math. 20, American Math. Soc., 1971, 415-440.
5. M. A. Morrison and J. Brillhart, "A method of factorization and the factorization of F7 ", Mathematics of Computation 29 (1975), 183-205.
6. C. Pomerance, "Analysis and comparison of some integer factoring algorithms",
in Computational Methods in Number Theory (edited by H. W. Lenstra, Jr. and R. Tijdeman), Math. Centrum Tract 154, Amsterdam, 1982, 89-139
7. J. M. Pollard, "A Monte Carlo method for factorization", BIT 15 (1975), 331-334.
8 R. S. Lehman, "Factoring large integers", Mathematics of Computation 28 (1974),637-646.
9. M. Voorhoeve, "Factorization", in Studieweek Getaltheorie en Computers (edited by J. van de Lune), Math. Centrum, Amsterdam, 1980, 61-68.
10. Junod, Pascal. "Cryptographic Secure Pseudo-Random Bits Generation: The Blum-Blum-Shub Generator." August 1999. Available: http://www.win.tue.nl/~henkvt/boneh-bbs.pdf
11. H. W. Lenstra, Jr., "Factoring integers with elliptic curves", Ann. of Math. (2) 126 (1987), 649-673.
12. "General number field sieve." From Wikipedia, an online encyclopedia. Available: http://en.wikipedia.org/wiki/GNFS
13. Wesstein, Eric W. "RSA Encryption." From Mathworld, an online encyclopedia. Available: http://mathworld.wolfram.com/RSAEncryption.html
14. R. P. Brent, "Some integer factorization algorithms using elliptic curves", Australian Computer Science Communications 8 (1986), 149-163.
15. Housley et al. "RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile.". Available: http://www.faqs.org/rfcs/rfc2459.html