East Asian Math. J. 19 (2003), No. 1, pp 81–89

A PRIME FACTORIZATION ALGORITHM IN ACTIONSCRIPT

TAI SUNG SONG

ABSTRACT There are many algorithms for factoring integers The trial division algorithm is one of the most efficient algorithms for factoring small integers (say less than 10,000,000,000) For a number n to be factored, the runtime of the trial division algorithm depends mainly on the size of a nontrivial factor of n In this paper, we create a function named factors that can implement the trial division algorithm in ActionScript and using the factors function we construct an interactive Prime Factorization Movie and an interactive GCD Movie

1. Introduction

Each positive integer greater than 1 can be expressed as a unique product of prime numbers. There are many algorithms for finding a nontrivial factor of a composite integer. The most useful algorithms fall into one of two classes: general purpose algorithms and special purpose algorithms [1]. The general purpose algorithms (such as the quadratic sieve and the number field sieve) have an expected runtime that depends mainly on the size of the number n being factored. The special purpose algorithms (such as trial division, Pollard rho method, and the elliptic curve method) have an expected runtime that depends mainly on the size of a nontrivial factor of n.

Received May 12, 2003. Revised May 23, 2003
2000 Mathematics Subject Classification: 97U70.
Key words and phrases: standard form, trial division, ActionScript.
This work was supported by Pusan National University Research Grant

The most straightforward method of factoring integers is trial division, where one simply tries to divide by each prime up to the square root of the number n to factor. The number of primes less than or equal to an integer n is denoted by $\pi(n)$, the prime counting function. According to the prime number theorem [2], we have $\pi(n) \approx \frac{n}{\log n}$. Since a composite number n has at least one factor less than or equal to $\sqrt{n}$, factoring using trial division takes approximately $\frac{\sqrt{n}}{\log \sqrt{n}}$ operations, in the worst case. For many composite numbers trial division is therefore infeasible as factoring method. For most numbers it is very effective, however, because most numbers have small factors. 88% of all positive integers have a factor less than 100, and almost 92% have a factor less than 1000 [3]. In fact, the trial division algorithm is one of the most efficient algorithms for factoring small integers (say less than 10,000,000,000).

In this paper, we create a function named factors that can implement the trial division algorithm in ActionScript and using the factors function we construct an interactive Prime Factorization Movie and an interactive GCD Movie. ActionScript, the scripting language of Macromedia Flash MX, is an object-oriented scripting language. The ActionScript syntax and style closely resemble that of JavaScript([4], [5]).

2. A Prime Factorization Algorithm

If a number n is composite, it will have a factor less than or equal to $\sqrt{n}$. Through an algorithm, like the Sieve of Eratosthenes, generate a list of the primes less than or equal to $\sqrt{n}$. The Sieve of Eratosthenes finds all prime numbers less than or equal to an integer n: Make a list of all the integers less than or equal to n and strike out the multiples of all primes less than or equal to $\sqrt{n}$, then the numbers that are left are primes. Then, try dividing n by each of these primes. Once one prime factor p is found, do trivial division on n/p. Repeat until all prime factors are found.

If we does not want to generate and store the list of prime num-

bers, we can first try dividing n by 2. Then, try dividing n by each of the odd integers less than or equal to $\sqrt{n}$. This algorithm saves the trouble of generating the primes without increasing the number of trial divisions considerably.

In this section, we establish a prime factorization algorithm in ActionScript using the latter trial division method.

Since the standard form of the integer 72 is $2^3 \cdot 3^2$, the array of prime factors of 72, counting multiplicities, is [2, 2, 2, 3, 3], and the array of distinct prime factors of 72 is [2, 3]. Using the trial division method decsribed above, we define the factors (n) function that returns the array of prime factors of n, counting multiplicities. The algorithm of the factors function can be written in ActionScript code as follows.

```
function factors (n) {
var A = new Array();
var p = 2,
while (n > 1) {
   if (n%p == 0) {A.push(p); n = n/p;}
   else if (p == 2) {p = 3;}
   else if (p > Math sqrt(n)) {p = n;}
   else {p = p + 2;}
}
return A;
}
```

Using the factors (n) function, we define the primeFactors (n) function that returns the array of distinct prime factors of n. The algorithm of the primeFactors function can be written in ActionScript code as follows.

```
function primeFactors (n) {
A = factors (n);
var s = 1;
b = A.length;
```

```
C = new Array ();
C[0] = A[0];
for (var i = 1; i < b; i++) {
   if(A[i] != A[i-1]) {C[s] = A[i]; s = s+1;
   } else {continue;}
}
return C;
}
```

Let a and b be positive integers. If b^k is a factor of a, then $b^k \leq a$ or $k \leq \frac{\log a}{\log b}$. Using this result, we can define the power (a, b) function that returns a largest integer k such that b^k is a factor of a. For example, power (121176, 3) is 4, since $121176=2^3 \cdot 3^4 \cdot 11 \cdot 17$. The algorithm of the power function can be written in ActionScript code as follows.

```
function power(a, b) {
pow = Math.pow;
ceil = Math.ceil;
log = Math.log;
limit = ceil(log(a)/log(b));
for (var k = 1; k <= limit; k++) {
   if(a % pow(b, k) != 0) {break;}
}
return k-1;
}
```

For the standard form $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ of an integer n, we define the powerArray (n) function that returns the array $[e_1, e_2, \cdots, e_k]$, using the primeFactors (n) and power (a, b) functions. For example, powerArray (121176) is the array [3, 4, 1, 1], since $121176=2^3 \cdot 3^4 \cdot 11 \cdot 17$. The algorithm of the powerArray function can be written in ActionScript code as follows.

```
function powerArray (n) {
```

```
A = primeFactors (n);
b = A.length;
C = new Array ();
for (var i = 0; i < b; i++) {
   C[i] = power (n, A[i]);
}
return C;
}
```

Using the primeFactors (n) and powerArray (n) functions, we now define the standardForm (n) function that returns the standard form of an integer n. The algorithm of the standardForm function can be written in ActionScript code as follows.

```
function standardForm (n) {
A = primeFactors (n);
B = powerArray (n);
c = A.length;
s = " ";
for (var i = 0; i < c; i++) {
   s = s + A[i] + "^" + B[i] + "   ";
}
return s;
}
```

Using five functions (factors, primeFactors, power, powerArray, and standardForm functions) defined above, we construct the primeFactor.fla file that can generate an interactive Prime Factorization Movie. First, we insert ActionScript codes of five functions in the first frame of the movie. Next, we create a graphical user interface of the movie that contains an Input Text field, a Dynamic Text field, a button instance, and other suitable objects. We declare the variable name of the Input Text field as box1 and the variable name of the Dynamic text field as box2. Finally we assign the following ActionScript code to the button instance.

```
on (release) {
   n = Number(box1);
   s = standardForm (n);
   box2 = s;
}
```

If we publish the primeFactor.fla file, then primeFactor.swf and primeFactor.html files are generated. This completes the construction of an interactive Prime Factorization Movie. If we enter an integer n to be factored in the Input Text field of the interactive Prime Factorization Movie and click the button, then the standard form of the integer n is displayed in the Dynamic Text field.

3. A GCD Algorithm

In this section, we construct an interactive GCD Movie that computes the greatest common divisors (factors) of two integers.

The sort(); method is the lexicographic order sort function. We define a compare function named order such that the sort(order); method sorts the elements of an array in numerical order. Note that 30 comes before 5 in lexicographic order, but in numerical order 5 comes before 30. The following code is an ActionScript code that defines the order function.

```
function order (x, y) {return x - y;}
```

Using primeFactors (n) and order (x, y) functions, we define the concatFactors (m, n) function that returns the array which consists of all distinct prime factors of m or n. For example, concatFactors (39546, 47320) is the array [2, 3, 5, 7, 13], since $39546 = 2 \cdot 3^2 \cdot 13^3$ and $47320 = 2^3 \cdot 5 \cdot 7 \cdot 13^2$. The algorithm of the concatFactors function can be written in ActionScript code as follows.

```
function concatFactors (m, n) {
```

```
var A = primeFactors (m);
var B = primeFactors (n);
C = new Array ();
C = A.concat(B);
C = C.sort(order);
var k = C.length;
D = new Array ();
D[0] = C[0];
for (var i = 1; i < k; i++) {
   if(C[i] != C[i-1]) {D.push(C[i]);
   } else {continue;}
}
return D;
}
```

Let m = 39546 and n = 47320. To compute the gcd of m and n, we compare the prime factors of m and n Note that $m = 2 \cdot 3^2 \cdot 13^3$, $n = 2^3 \ 5 \cdot 7 \cdot 13^2$, and concatFactors = [2, 3, 5, 7, 13]. Since 2 is a factor of both m and n, and the smallest exponent on 2 is 1, so 2^1 is a factor of gcd (m, n). Since 3 is a factor of m but not n, 3^0=1 is a factor of gcd (m, n) Similarly, 5^0=1 and 7^0=1 are factors of gcd (m, n). Also, 13 is a factor of both m and n, and the smallest exponent on 13 is 2, so that 13^2 is a factor of gcd (m, n). Consequently, we obtain the following result.

$$\gcd(m, n) = 2^1 \cdot 3^0 \cdot 5^0 \cdot 7^0 \cdot 13^2 = 338.$$

Using concatFactors and power functions, and the method described above, we now define the gcd (m, n) function that returns the gcd of two integers. The algorithm of the gcd function can be written in ActionScript code as follows.

```
function gcd (m, n) {
var A = concatFactors (m, n);
var k = A.length,
B = new Array ();
```

```
for (var i = 0; i < k; i++){
   B[i] = power(m, A[i]);
}
C = new Array ();
for (var i = 0; i < k; i++) {
   C[i]=power(n, A[i]);
}
D = new Array ();
for (var i = 0; i < k; i++) {
   D[i]=Math.min(B[i], C[i]);
}
var s = 1;
for (var i = 0; i < k; i++) {
   s = s* pow(A[i], D[i]);
}
return s;
}
```

Using the same method of constructing the interactive Prime Factorization Movie, we can construct an interactive GCD Movie, the gcd.fla file. We insert ActionScript codes of six functions (factors, primeFactors, power, order, concatFactors, and gcd functions) defined above in the first frame of the movie, and create a graphical user interface of the movie that contains two Input Text fields, a Dynamic Text field, a button instance, and other suitable objects. We assign the following ActionScript code to the button instance, where box1 and box2 are the variable names of the Input Text fields, and box3 is the variable name of the Dynamic text field.

```
on (release) {
   m = Number(box1);
   n = Number(box2);
   s = gcd (m, n);
   box3 = s;
}
```

If we publish the gcd.fla file, then gcd.swf and gcd.html files are generated. This completes the construction of an interactive GCD Movie. For example, if we enter two integers 1156377601, 1048012963 in the Input Text fields of the interactive GCD Movie and click the button, then 488129, the gcd of two integers, is displayed in the Dynamic Text field.

REFERENCES

[1] R P Brent, *Recent progress and prospects for integer factorisation algorithms, Lecture Notes in Computer Science, Vol. 1858*, Springer Verlag, Berlin, 2000

[2] G H Hardy and W M Wright, *An Introduction to the Theory of Numbers, 5th ed* , Oxford University Press, Oxford, 1979

[3] A K. Lenstra, *Integer Factoring*, Designs, Codes and Cryptography **19(2/3)** (2000), 101–128

[4] Macromedia, *Flash MX ActionScript Dictionary*, Macromedia, San Francisco, CA , 2002

[5] Netscape, *Core JavaScript Reference*, Netscape, Mountain View, CA., 1998.

Department of Mathematics Education
Pusan National University
Pusan 609-735, Korea
E-mail: tssong@pusan.ac.kr