AVALG13

Integer Factorization

November 7, 2013

Carl Eriksson **carerik@kth.se**
Viktor Mattsson **vikmat@kth.se**

Contents

1 Introduction

1.1 Statement of collaboration

1.2 Problem statement

Prime numbers are a quite interesting set of numbers. They share the special trait that they are only divisible by one or themselves, unlike non-primes. However, every non-prime number can be written as a series of prime number which is useful when computing data with large integers as it allow dividing a big problem into several smaller ones. Finding this series of primes or more commonly, finding the factors of these large integers is however no simple matter and efficient algorithms are needed.

1.3 Definitions

- N - Number to be factorized
- GCD - Greatest common divisor
- Kattis - Code “judge” used by KTH to test code

2 Algorithms

2.1 Used

2.1.1 Trial division

Trial division is one of the most inefficient algorithms for factorization of large integers. It is however one of the most efficient algorithms for factoring small integers and it also the easiest algorithm to perceive. The trial division algorithm only have a few steps. It begins with finding a set of prime-numbers using some algorithm such as the sieve of eratosthenes then iteratively check if N divided by any of these primes grants no remainder. If it does not grant any remainder the prime used in the division is a factor. The algorithm will continue checking this until N becomes one. A simple optimization of trial division is to never check primes greater than the square root of N. Considering that if $N = a * b$ and if a is greater than $\sqrt{N}$ then b must be smaller than $\sqrt{N}$. So for every factor greater than $\sqrt{N}$ there has to be one factor smaller than $\sqrt{N}$ thus if we check for factors up to $\sqrt{N}$ we will find a factor unless N is already prime.

Algorithm 1 Trial division

```
function TRAIL_DIVISION($N$)
    if $N = 1$ then
        return
    end if
    $list\_of\_primes = sieve\_of\_eratosthenes(\sqrt{N})$
    prime_factors = {}
    for all $prime$ in $primes$ do
        while $n \bmod prime = 0$ do
            $prime\_factors.add(prime)$
            $n = n/p$
        end while
    end for
    if $n$ not equal to 1 then
        return $prime\_factors.add(n)$
    end if
    return $prime\_factors$
end function
```

2.1.1.1 Pseudocode:

2.2 Considered