Integer Factorization

Diana Gren
Jonas Carlsson

DD2440 Advanced Algorithms
Stefan Nilsson

Contents

Chapter 1

Background

Factorize integers into prime numbers is considered a difficult problem. There is algorithms that solve the problem, but they can in some cases be very time consuming. An example is a product of two large prime numbers, which is very difficlt to factorize. This is the key in RSA cryptography, which completely relies on the fact that factoring products of large prime numbers is incredibly difficult.

1.1 Algorithms

As mentioned above, there are some algorithms that solve the problem, of which two of them will be discussed in this section.

1.1.1 Pollard Rho

Pollard Rho is a fairly easy algorithm to both understand and implement, and it has an acceptable performance. The algorithm was invented by John Pollard in 1975, and its speciality is to factor composite numbers with small factors.

1.1.2 Quadratic Sieve

The Quadratic Sieve is a little more complicated.

Chapter 2

Approach

We decided to do the programming in C++. We both had limited knowledge in the language and wanted to learn more, as well as handling I/O for Kattis felt like a much easier task in C++ than in Java.

Our approach for solving the problem was to implement the Pollard Rho algorithm. We did some research on the Quadratic Sieve, but ended up with Pollard Rho mostly because it is an easy algorithm, and at the same time it can perform quite well. We tested it towards Kattis, and tried to optimize it in different ways and in this chapter, we will discuss more into detail what was tested.

2.1 Pollard Rho

2.1.1 Implementation

We quickly realized that we needed the program to stop looking for factors if it was taking too long, so we introduced a cut off limit. The cut off limit is a limit for the maximum number of iterations the algorithm is allowed to do, and gives the answer "fail" if it did not succeed in fatorizing the given number.

The scores could be improved just by increasing the cut off limit so that the program would run for as close to 15 seconds as possible, since it obviously had time to evaluate more numbers.

```
f(z, N)
        return z*z % N
function pollard(N)
        x := random(N)
        y := x
```

```
        prod := 1

        for i := 1 to infinity do
                x = f(x, N)
                y = f(f(y, N), N)

                if (x - y = 0) then
                        prod = prod *(x-y) % N
                        d := gcd(N, prod)

                if d > 1 and d < N then
                        return d

                if i >= CUT_OFF_LIMIT then
                        return 0
        end for
end function
```

2.1.2 Optimizations

We recognized that since there is a 50/50 chance that the given number is even, it could be a good idea to check that before running the algorithm, and thereby eliminate calculation time. So we started off by introducing a check for even numbers, and just return true if the condition was satisfied.

After reading more on Pollard Rho, we found that finding the greatest common divisor is quite expensive to do, hence we do not want to do that so often. This made us multiply the value a certain number of times before entering the gcd, hoping to improve the algorithm.

2.2 Pollard Rho with Brent

By implementing the Brent cycle finding algorithm, we hoped to improve the Pollard Rho algorithm and receive higher scores.

2.2.1 Implementation

2.2.2 Problems

Chapter 3

Results

Cut off limit	Score	Time [s]
130	12	1.2
1000	15	2.4
10000	32	12.1
12500	39	14.20

Chapter 4

Conclusion