Factoring

Ibrahim Said - isaid@kth.se
Kwabena Asante-Poku kwap@kth.se

November 2016
DD2440 - Advanced Algorithms
Kattis id: 1555738
Score: 64

1 Introduction

The main purpose of this paper is to analyze different methods for solving the prime factorization problem. This is known to be a difficult problem when the input integer becomes large enough. This is of interest since this difficulty is what many cryptographic protocols are based on.

1.1 Problem formulation

Given an integer $i \geq 2$ we want to decompose it into a set $\{f_1, f_2, f_3 \dots f_n\}, f_i \in \mathbb{Z}$ so that $f_1 * f_2 * f_3 * \cdots * f_n = i$ and each f_i is a prime. This means that if i is prime the resulting set will simply be $\{i\}$.

In our specific task the problem is given by the KTH Kattis System as a series of test cases under the problem *Factoring* (problem id: kth.avalg.factoring). Input is given as integers larger than 1 and not more than 100 bits. Each test case is allowed to run a maximum of 15 s. The algorithm is then tested by outputting the prime factorization for each number. A score is given between 0 and 100 based on the relation of successfully factorized integers and total integers. As an added option due to the time constraint it can also choose to output a fail message to indicate that the problem could not be solved in time.

1.2 Language

Our language of choice to solve this task was C++ due to its extensive libraries and low level capabilities. Since the integers used as input for our task can make use of up to 100 bits the standard integers are not guaranteed to be able to represent a 100 bit integer without overflowing. This was circumvented by using GNU's Bignum library GMP.

1.3 Approach

For our initial attempt we implemented Fermat's factorization method since it was one of the more trivial options. However, we noticed that the running time of the algorithm was sub-par for larger numbers which led us to implement a more suitable algorithm.

The second algorithm we used was Pollard's Rho algorithm. Which is fairly simple and relies on a cycle-finding algorithm and a psuedo-random number generator. It was chosen due to the fact that it is probabilistic which would most likely improve the running time a great deal compared to Fermat's factorization method. As expected, there was a notable difference in the time needed to decompose the input integer and each subsequent resulting in more problems being solved in the given time frame.

In an attempt to further improve the results we added an initial routine which first decides if the integer is most likely prime and if not, does a trial division with the first n primes in a sequence.

1.4 Fermat's factorization algorithm

It is well known that difference of two squares can be factorized as following:

$$a^2 - b^2 = (a + b)(a - b) \tag{1}$$

This equation lies in the core of Fermat's algorithm. Given an odd integer n which we want to factorize we introduce an integer c and subtract c^2 by n to get a value d. If $d_r = \sqrt{d}$ is also an

integer we can write the equation $n = c^2 - d_r^2$ which due to 1.4 gives the decomposition of n as $n = (c + d_r)(c - d_r)$.

Since this basic method only gives a general decomposition (not necessarily prime as required by our problem formulation) we extended it by running the algorithm on the subsequent components of the decomposition and marking a component as prime if the algorithm could not decompose it. Further, since the algorithm takes an odd integer we ran routine before each decomposition to see if the integer to decompose n_i was even. If it was even we simply decomposed it directly with the components $n_i/2$ and 2 with the exception that n_i was 2 as 2 is a prime.

1.4.1 Pseudocode

Algorithm 1 Fermat's Factorization Method

```
1: procedure FERMAT(N)
2:     N is odd
3:     c = ceil(sqrt(N))
4:     d = c^2 − N
5:
6:     while √d not positive integer i in A do
7:         c+ = 1
8:         d = c^2 − N
9:     end while
10:    f1 = c + √d
11:    f2 = c − √d
12:    Return f1, f2
13: end procedure
```

1.4.2 Implementation

The data structures used for our implementation were two linked lists were one was used as a FIFO-queue for integers to be decomposed and the other was used for storing and outputting the results. The choice of using a linked list was based on the feature that there will be no need to reallocate memory for the whole structure when dequeuing and enqueuing which may happen in a an array.

1.5 Pollard's rho algorithm

Pollard's Rho algorithm is based on several concepts which gives it an edge over Fermat's algorithm. Given an integer to factorize n, introduce two integer variables x_1, x_2 and function $f(x)$. Set both of the variables to the same value $0 \leq val < n$ and set the function to $ax^2 + c$ where a and c are constants. Set x_1 to $f(x_1) \mod n$ and x_2 to $f(f(x_2)) \mod n$. Calculate $\gcd(|x_1 - x_2|, n)$. Since we know there is a $|x_1 - x_2|$ which will give us a $gcd > 1$ it makes use of the birthday paradox which implies that there is a roughly 50% chance of finding the correct x_1 and x_2 after $\sqrt{n}$ iterations.

1.5.1 Pseudocode

Algorithm 2 Pollard's rho Algorithm

```
1: procedure POLLARD(N)
2:     input : x
3:     x1 = f(x)
4:     x2 = f(f(x))
5:     d = 1
6:
7:     while while d = 1 do
8:         x1 = f(x1)
9:         x2 = f(f(x2))
10:        d = gcd(|x1 − x2|, N)
11:    end while
12:    if d ! = N then
13:        Return d
14:    else
15:        Return "fail"
16:    end if
17: end procedure
```

1.5.2 Implementation

The data structures used for the Pollard's Rho implementation were, as in Fermat's, also two linked lists used in the same manner where one was used as a FIFO-queue for integers to be decomposed and the other was used for storing and outputting the results.

2 Result

Factoring using Fermat's factorization algorithm with an iteration limit of 600000 for each decomposition gave a score of 2 (id: 1559276) in 14.30 s. Factoring numbers using Pollard's rho algorithm with a non-changing number generator and a max cycle limit of 100000 yielded the score 37 (id: 1553361) in 7.68 s. Increasing the cycle limit to 200000 resulted in a score of 45 (id: 1555407) in 14.59 s. Further, adding a trial division of size 1000 (while lowering cycle limit to 160000) increased the score to 64 (id:1555566).

3 Conclusion

The poor performance of Fermat's algorithm can be attributed to the excessive runtime needed as the numbers become larger. This is where Pollard's Rho algorithm outperforms Fermat's. Due to the fact that the randomness makes it more suitable for larger numbers. A setback with Pollard's is that it may not be able to find a solution at all depending on the random number generator but since the problem is restricted by time in our task it is generally still a more preferable method as Fermat's method times out for large enough numbers.

4 Final words

Kwabena implemented the Pollard rho algorithm while Ibrahim implemented Fermat's method. Ibrahim also implemented the tool code which contains multiple functions of use when implementing the algorithms. This has of course been done by helping each other and The report have as well been fairly divided amongst both.
We believe that we have fulfilled the criteria for E.