Index Calculus

Simone D'Aniello

TEN 17/18

1 Notations

p = Prime number
g = Primitive root
B = Smoothness bound
b = Argument such that $log_g b = exp$

2 Pseudocode

```
set the factor base of primes p_k in [2, B]
initialize the array of vectors of exponents vexp
while vexp is not filled enough
    choose x random in interval [0, p-2]
    compute y = g^x mod p
    if y is B-smooth
        y mod p = (-1)^e_i0 (2)^e_i1 ....(p_k)^e_ik
        v_i = (e_i0, e_i1, ..., e_ik)
        add v_i to vexp
    if vexp is filled enough
        if not span back to 2
    else
        compute log_g p_i for each prime in the factor base
        while g^beta * b is not B-smooth
            choose beta random in the interval [0, p-2]
        g^beta * b mod p = (p_1)^f1 (p_2)^f2 ....(p_k)^f_k
        find log_g b using the relation beta + log_g b = f_1 log_g p_1 + ... + f_k log_g p_k
        return exp = log_g b
```

3 Implementation

```
1 set the factor base of primes p_k in [2, B]
```

In the code is used the library GMP for the computation of the discrete logarithm for big numbers. For the initialization of the factor base is used the function `mpz_probab_prime_p` that performs some trial division and then x Miller-Rabin probabilistic primality tests, where x is set to 15.

```
2 initialize the array of vectors of exponents vexp
3 while vexp is not filled enough
```

let k be the cardinality of the Factor Base. *vexp* is considered filled enough if contains $k + k/7$ vectors of exponents

```
4     choose x random in interval [0, p-2]
5     compute y = g^x mod p
```

The first thing the program does is discovering the number of processors *nprocs* in the PC. Then it creates *nprocs* threads with unique identifiers. This identifiers are used as seeds for the random number generator so each thread can fill *vexp* with different vectors.

```
6     if y is B–smooth
7         y mod p = (-1)^e_i0 (2)^e_i1 ....(p_k)^e_ik
8         v_i = (e_i0, e_i1, ..., e_ik)
9         add v_i to vexp
```

To check if a number x is B-Smooth, x is divided in factors iterating Pollard's Rho algorithm. if x is too big to be factored the program uses the extended Euclidean algorithm to compute $s * x + t * p = z$. x is represented as $\frac{z}{s}$ and is B-smooth if s and z are. If so, $x = (f_{z1} - f_{s1})log_g p_1 + (f_{z2} - f_{s2})log_g p_2 ...$

```
10    if vexp is filled enough
11        if not span back to 2
12    else
13        compute log_g p_i for each prime in the factor base
14        while g^beta * b is not B–smooth
15            choose beta random in the interval [0, p-2]
16        g^beta * b mod p = (p_1)^f1 (p_2)^f2 ....(p_k)^f_k
17        find log_g b using the relation beta + log_g b = f_1 log_g p_1 + ... + f_k log_g p_k
18        return exp = log_g b
```

4 Code Structure

indexCalculus.c	Is the core of the application. Contains the representation of the pseudocode in C language
matrixOperations.c	In this file there are functions to execute the Gauss-Jordan algorithm for solving systems of linear equations
factorOperations.c	Contains the function to check if a number is B-smooth and the implementation of the Pollard's rho algorithm
arrayFunctions.c	Used to initilize, free and compute operations on arrays
general.c	Used for the print functions, the computation of $log_e B$ and all the primes in the factor base

5 How to run the code

The next code represents a default makefile. The program indexCalculus doesn't use any parameter written using the command line interface. The four parameters P, B, ROOT and WANTED are initialized modifying the makefile.

```
1  TARGET = indexCalculus
2  LIBS = -lgmp -lm -lpthread
3  CC = gcc
4  CFLAGS =  -Wall -Wextra -DB=0 -DP=\"5915587277\" -DROOT=2 -DWANTED=42 -
       DVERBOSE=1  -O2 -o -lgmp -lpthread -lm
5
6  .PHONY: default all clean
7
8  default: $(TARGET)
9  all: default
10
11 OBJECTS = $(patsubst %.c, %.o, $(wildcard *.c))
12 HEADERS = $(wildcard *.h)
13
14 %.o: %.c $(HEADERS)
15   $(CC) $(CFLAGS) -c $< -o $
16
17 .PRECIOUS: $(TARGET) $(OBJECTS)
18
19 $(TARGET): $(OBJECTS)
20   $(CC) $(OBJECTS) -Wall $(LIBS) -o $
21
22 clean:
23     -rm -f *.o
24     -rm -f \$(TARGET)
25      clear
```

- B is the smoothness bound. The default value is 0. If B is 0 the program will compute the optimal value for the inserted prime. Optimal B is computed as $B = e^{\sqrt{\frac{lnp*lnlnp}{4}}}$
- P is the prime of $\mathbb{Z}_P$ written as a string

- ROOT is a primitive root for the prime P
- WANTED is the b such that $log_{ROOT} WANTED = output$

In order to run the code you have to

1. change the parameters in the makefile
2. execute *make clean* in the command line interface
3. execute *make* in the command line interface
4. execute *./indexCalculus*

6 Results

P	digits	seconds
663424523745421	15	3
48112959837082048697	20	40
40206835204840513073	20	68
200000000000000002487	21	46
123456789878987654321	21	47
666276812967623946997	21	60
200000000000000002127	22	118
19171311753235711131719	23	213
22360679774997896964091	23	140
200000001566924774852 87	23	157
20000000014296355084 4099	24	355