Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Криптография»

Студент:	Тояков А. О.
Преподаватель:	Борисов А. В.
Группа:	М8О-307Б-18
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Вариант №2

Разложить каждое из чисел представленных ниже на нетривиальные сомножители.

Первое число:

n1 =
3523581180791504931870993551416295271017491061679972555096190205283337223522 17

Второе число:

n2 =
1695128485402083763773247025508607781296883851800934596605324477902989896723 90
0984413142336870385225437965243629326745116590849908770944614057690683052539 80
1654819522761512642822701693074249824513493644688844526263633663327921066974 98
3001545042891090435383147221714908515772020029364695158378468844726857013205 55
9546752704709817118834528761529676361607229919430317377276744622348039645465 22
34970667881341234171270319084202556797982227882925483764275373954664 9159

Выходные данные:

Для каждого числа необходимо найти и вывести все его множетели - простые числа.

1 Описание

Процесс разложения числа на его простые множетели называется факторизацией. Для решения этой задачи существует множество алгоритмов, позволяющих находить множетели, используя свойства простых чисел.

Для решения задачи я выбрал алгоритм ρ-Полларда[1], как один из наиболее простых и эффективных. Однако эффективен он, как я узнал позже, для чисел меньшего порядка, чем в моем задании. Моя программа работала полсуток, после чего было решено прервать выполнение и искать другие методы .

Для чисел, десятичнных знаков меньше 100 эффективен Метод квадратичного решета, реализация которого довольно трудоемка, поэтому, так как преподаватель не ограничил нас в выборе средств для решения задачи, я решил прибегнуть к готовому решению, реализованному профессионалами - программному продукту **msieve**[2], который реализует в себе целый ряд алгоритмов факторизации с общим названием Общий метод решета числового поля. Этот метод считается одним из самых эффективных современных алгоритмов факторизации. Он справился с поставленной задачей для 1-го числа примерно за 2 минуты,.

Однако 2 число имеет более 400 квадратичных знаков, факторизация которого на обычном компьютере за разумное время практически невозможна ни одним из ныне существующих алгоритмов. Однако была дана подсказка, что один из множетелей этого числа определяется к наибольший общий делитель с одним из чисел другого варанта. Поэтому я быстро написал программу, пребирающую все числа других варантов, определяющую их *НОД* с числом моего варанта и выводящий его, если он > 1. Второе же число определяется как результат деления числа моего варанта на *НОД* (по свойству делителя). Для работы с большими числами и отыскания делителя в этой программе я использовал библиотеку **gmp**[3].

2 Исходный код

Моя реализация алгоритма ρ - *Полларда* на языке C++, описание которого я взял из [1].

```
#include <iostream>
#include <string>
#include <gmpxx.h>
#include <vector>
#include <ctime>


class po_Polard{
public:
    po_Polard(const mpz_class& num);
    po_Polard(const std::string& num);
    mpz_class get_factor();
private:
    mpz_class factor_of_num(mpz_class& num);
    void Polard_GCD(mpz_class& ans, mpz_class& x, mpz_class& y);
    void Polard_MOD(mpz_class& ans, mpz_class& x, mpz_class& y);
    void Polard_ABSOLUTE(mpz_class& ans, mpz_class& x, mpz_class& y);
    mpz_class number;
};

mpz_class po_Polard::factor_of_num(mpz_class& num){
    mpz_class x, y, ans, absolute;
    unsigned long long i = 0, stage = 2;
    x = (rand() % (number - 1)) + 1;
    y = 1;
    Polard_ABSOLUTE(absolute, x, y);
    Polard_GCD(ans, num, absolute);
    while(ans == 1){
        if(i == stage){
            y = x;
            stage <<= 1;
        }
        absolute = x * x + 1;
        Polard_MOD(x, absolute, num);
        ++i;
        Polard_ABSOLUTE(absolute, x, y);
        Polard_GCD(ans, num, absolute);
    }
    return ans;
}

mpz_class po_Polard::get_factor(){
    return factor_of_num(number);
}
```

```cpp


po_Polard::po_Polard(const mpz_class& num){
    srand(time(0));
    number = num;
}

po_Polard::po_Polard(const std::string& str){
    srand(time(0));
    number = str;
}

void po_Polard::Polard_ABSOLUTE(mpz_class& ans, mpz_class& x, mpz_class& y){
    x -= y;
    mpz_abs(ans.get_mpz_t(), x.get_mpz_t());
    x += y;
}


void po_Polard::Polard_GCD(mpz_class& ans, mpz_class& x, mpz_class& y){
    mpz_gcd(ans.get_mpz_t(), x.get_mpz_t(), y.get_mpz_t());
}

void po_Polard::Polard_MOD(mpz_class& ans, mpz_class& x, mpz_class& y){
    mpz_mod(ans.get_mpz_t(), x.get_mpz_t(), y.get_mpz_t());
}

using namespace std;

int main(){
    std::string number = "
        3523581180791504931870993551416295271017491061679972555096190205283337223522217"
        ;
    po_Polard polard(number);
    std::cout << "Factor: " << polard.get_factor() << endl;
    return 0;
}
```

3 Консоль

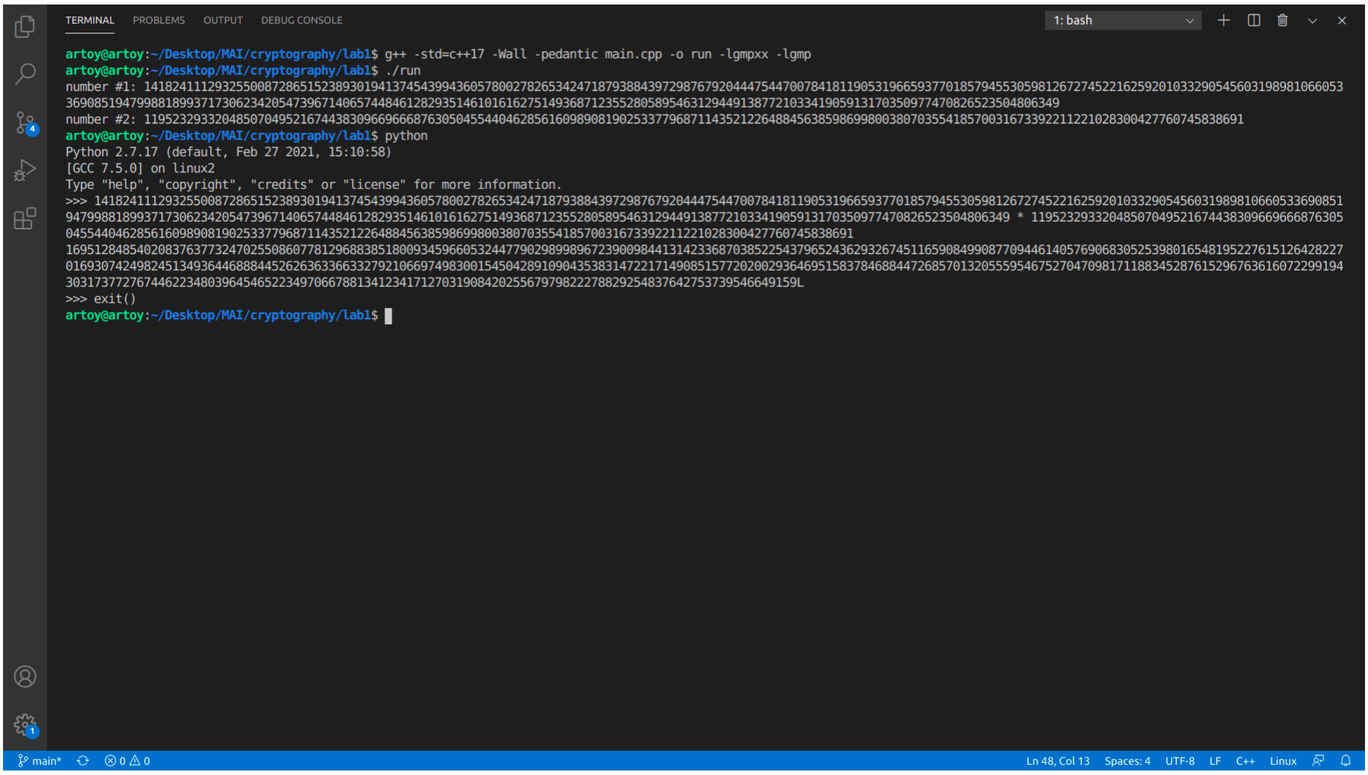
```
artoy@artoy:~/Math/msieve$ ./msieve 35235811807915049318709935514162952710174910667997

sieving in progress (press Ctrl-C to pause)
39370 relations (19984 full + 19386 combined from 216570 partial), need 39162
sieving complete, commencing postprocessing

artoy@artoy:~/Math/msieve$ cat msieve.log
Sun Apr 11 18:51:43 2021
Sun Apr 11 18:51:43 2021
Sun Apr 11 18:51:43 2021  Msieve v. 1.54 (SVN 1040)
Sun Apr 11 18:51:43 2021  random seeds: e95ed60d 0df04467
Sun Apr 11 18:51:43 2021  factoring 35235811807915049318709935514162952710174910616799
Sun Apr 11 18:51:43 2021  no P-1/P+1/ECM available, skipping
Sun Apr 11 18:51:43 2021  commencing quadratic sieve (78-digit input)
Sun Apr 11 18:51:43 2021  using multiplier of 13
Sun Apr 11 18:51:43 2021  using generic 32kb sieve core
Sun Apr 11 18:51:43 2021  sieve interval: 12 blocks of size 32768
Sun Apr 11 18:51:43 2021  processing polynomials in batches of 17
Sun Apr 11 18:51:43 2021  using a sieve bound of 999269 (39066 primes)
Sun Apr 11 18:51:43 2021  using large prime bound of 99926900 (26 bits)
Sun Apr 11 18:51:43 2021  using trial factoring cutoff of 27 bits
Sun Apr 11 18:51:43 2021  polynomial 'A' values have 10 factors
Sun Apr 11 18:54:43 2021  39370 relations (19984 full + 19386 combined from 216570 pa
Sun Apr 11 18:54:43 2021  begin with 236554 relations
Sun Apr 11 18:54:43 2021  reduce to 56310 relations in 2 passes
Sun Apr 11 18:54:43 2021  attempting to read 56310 relations
Sun Apr 11 18:54:43 2021  recovered 56310 relations
Sun Apr 11 18:54:43 2021  recovered 47350 polynomials
Sun Apr 11 18:54:43 2021  attempting to build 39370 cycles
Sun Apr 11 18:54:43 2021  found 39370 cycles in 1 passes
Sun Apr 11 18:54:43 2021  distribution of cycle lengths:
Sun Apr 11 18:54:43 2021     length 1 : 19984
Sun Apr 11 18:54:43 2021     length 2 : 19386
Sun Apr 11 18:54:43 2021  largest cycle: 2 relations
Sun Apr 11 18:54:43 2021  matrix is 39066 x 39370 (5.6 MB) with weight 1162702 (29.53/
Sun Apr 11 18:54:43 2021  sparse part has weight 1162702 (29.53/col)
Sun Apr 11 18:54:43 2021  filtering completed in 3 passes
Sun Apr 11 18:54:43 2021  matrix is 28284 x 28347 (4.4 MB) with weight 917579 (32.37/
Sun Apr 11 18:54:43 2021  sparse part has weight 917579 (32.37/col)
```

```
Sun Apr 11 18:54:43 2021  saving the first 48 matrix rows for later
Sun Apr 11 18:54:43 2021  matrix includes 64 packed rows
Sun Apr 11 18:54:43 2021  matrix is 28236 x 28347 (2.6 MB) with weight 601688 (21.23/c
Sun Apr 11 18:54:43 2021  sparse part has weight 391154 (13.80/col)
Sun Apr 11 18:54:43 2021  commencing Lanczos iteration
Sun Apr 11 18:54:43 2021  memory use: 3.9 MB
Sun Apr 11 18:54:48 2021  lanczos halted after 448 iterations (dim = 28234)
Sun Apr 11 18:54:48 2021  recovered 17 nontrivial dependencies
Sun Apr 11 18:54:48 2021  p39 factor: 562068224324090847465842314308226273321
Sun Apr 11 18:54:48 2021  p39 factor: 626895637985717823820028254946860326577
Sun Apr 11 18:54:48 2021  elapsed time 00:03:05

artoy@artoy:~/Math/msieve$ python
Python 2.7.17 (default, Feb 27 2021, 15:10:58)
[GCC 7.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 562068224324090847465842314308226273321 * 626895637985717823820028254946860326577
352358118079150493187099355141629527101749106167997255509619020528333722352217L
>>> exit()
```



4 Ответ

Разложение первого числа:

- 562068224324090847465842314308226273321

- 626895637985717823820028254946860326577

Разложение второго числа:

- 14182411129325500872865152389301941374543994360578002782653424718793884397
  29876792044475447007841811905319665937701857945530598126727452216259201033
  29054560319898106605336908519479988189937173062342054739671406574484612829
  35146101616275149368712355280589546312944913877210334190591317035097747082
  6523504806349

- 11952329332048507049521674438309669666876305045544046285616098908190
  25337796871143521226488456385986998003807035541857003167339221122102
  8300427760745838691

5 Выводы

Благодаря превой лабораторной работе по курсу «Криптография», я напрямую познакомился с новой для себя темой - факторизацией больших чисел.

Эта работа сама по себе не является очень трудной для выполнения, однако я столкнулся со сложностью в выборе адекватного метода для поиска множетелей, ведь метод простого перебора, который превым мне пришел на ум, будет выполнять данную работу на моем компьютере больше времени, чем я смогу прожить, метод решета Эратофена, который я изучал ещё на первом курсе на парах по олимпиадному программированию для схожих задач, но с меньшими числами, также требует больше ресурсов, чем я могу представить для этой задачи, а метод Полларда, который я реализовал для этой задачи, не смог справиться с этой задачей в разумные сроки. Это стало мне уроком и показало насколько важно выбрать оптимальный алгоритм для решения задачи.

В целом, данная задача в курсе «Криптография» меня заинтересовала, так как благодаря ней я познакомился с несколькими новыми для меня алгоритмами и изучил много источников с интересной информацией.

Список литературы

[1] *Алгоритм ρ - Полларда*
URL: https://ru.wikipedia.org/wiki/Ро-алгоритм_Полларда (дата обращения: 08.04.2021).

[2] *Библиотека Msieve*
URL: https://github.com/radii/msieve (дата обращения: 09.04.2021).

[3] *Библиотека GMP*
URL: https://gmplib.org/ (дата обращения: 10.04.2021).