Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу Криптография

Студент:	В. А. Петросян
Преподаватель:	А. В. Борисов
Группа:	М8О-308Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Условие

Разложить каждое из чисел n1 и n2 на нетривиальные сомножители.
Вариант 15.
n1=383456614884902466726252731294544234658015390619372835826246625499154384118189,
n2=1416908444771934114327236064335695175033855568724514723276090909238902249450761163
1161792983700976377366095987469785396811908061750237394378249497902031141954472876211
9216205286391137003028125331158247702385902798481867910823926760076341189111357818193
8978341368763677855534685413427437290239276573078365437316891195505584463642669716112
9367283730885533859028643592189337506274405214704767741787934130977543281068768100090
83432628213288672194420754620920548851129

Метод решения

В олимпиадном программировании часто встречаются задачи требующие знаний из теории чисел. Задача разложения числа на простые множители одна из любимых тем составителей некоторых раундов на codeforces. Мне не раз приходилось с этим сталкиваться. Я понимаю, что асимптотика алгоритма очень важна при решении таких задач. Известные мне методы работают за $O(n * \lg n)$ и за $O(\sqrt{n})$. Их вполне хватает для решения контестов, но для выполнения данной лабораторной работы их использовать не эффективно.

Так как ограничений на иструменты не было первое число я разложил онлайн на сайте `https://www.alpertron.com.ar/ECM.HTM`
Второе число сайт не смог считать из-за ограничения на длину. Даже если не учитывать ограничение на длину сайт использовать всё равно нет смысла. Алгоритм в среднем потратил бы около 20 часов на факторизацию вторго числа. Меня это не особо обрадовало бы.

По совету старших коллег при факторизации второго числа был использован хак: первый множитель находится как НОД с числом другого варианта, а второй множитель – делением. Без длинной арифметики не обошлось. К счастью, она у меня сохранилась. Деление в моей длинке производится максимально эффективным алгоритмом
"q с крышечой"из книги Кнута.

Результат работы программы

```
n1 = 534 536246 117659 622855 155475 109020 504221 (39 digits) *
     717 363167 923504 642823 030581 565847 216209 (39 digits)

(base) vitya@Xubuntu18:~/Desktop/Petrosyan V.A./6 семестр/Криптография/Laba1/da6$ ls
bigint.cpp  bigint.hpp  bigint.o  input.txt  main.cpp  main.o  Makefile
(base) vitya@Xubuntu18:~/Desktop/Petrosyan V.A./6 семестр/Криптография/Laba1/da6$ make
g++ -Wall -pedantic -Werror -Wno-sign-compare -Wno-long-long -lm -o da6 main.o bigint.o
(base) vitya@Xubuntu18:~/Desktop/Petrosyan V.A./6 семестр/Криптография/Laba1/da6$ ./da6
```

0 RESULT 1

1 RESULT 1

2 RESULT 1

3 RESULT 1

4 RESULT 1

5 RESULT 1

6 RESULT 1

7 RESULT 1

8 RESULT 1

9 RESULT 1

10 RESULT 1

11 RESULT 1

12 RESULT 1

13 RESULT 1

14 RESULT 1

16 RESULT 139476625665968066568224614983908130916411561642255796653097148734670476110858310346843945046884204821224835911446227920807676218254025383087995289367050711834371298932142600299173740378514261264695250897646270832307124860459187526487820547916176260627474993626505280101101880100712184854128131522995503631439

17 RESULT 1

18 RESULT 1

19 RESULT 1

```
First
1394766256659680665682246149839081309164115616422557966530971487346704761108583103468
4394504688420482122483591144622792080767621825402538308799528936705071183437129893214
2600299173740378514261264695250897646270832307124860459187526487820547916176260627474
993626505280101101880100712184854128131522995503631439

Second
1015875196296533351595693081041988387023398869965302677760477005257173121893959513467
0761404904363773181670891345416776729531069936992072791918576655182711

ok
(base) vitya@Xubuntu18:~/Desktop/Petrosyan V.A./6 семестр/Криптография/Laba1/da6$
```

Листинг программного кода

```
#include "bigint.hpp"
using namespace std;

BigInt gcd(BigInt a,BigInt b){
    while ( !( a == 0 ) ){
        if( a < b ){
            BigInt tmp = a;
            a = b;
            b = tmp;
        }
        BigInt k = a / b;
        a = a - b*k;
        // cout << a << "\n";
    }
    return b;
}

int StingToInt(const string &str) {
    int tmp = 0;
    for(int i = 0; i < str.size(); ++i) {
        tmp = tmp * 10 + (str[i] - '0');
    }
    return tmp;
}

int main() {

    freopen("input.txt", "r", stdin);
    string strNum1;
    cin >> strNum1;
    BigInt num1(strNum1);

    for(int i = 0; i < 19; ++i){
        string strNum2;
        cin >> strNum2;
        BigInt num2(strNum2);
        BigInt res = gcd(num1,num2);
        if( i >= 15){
            cout << "\n\n" << i + 1;
        }
```

```
        else{
            cout << "\n\n" << i;
        }
        cout << "␣RESULT␣" << res << "\n\n";
    }
    BigInt sixteen("13947662566596806656822461498390813091641156164225579665
␣␣␣␣5309714873467047611085831034684394504688420482122483591144622792080767б
␣␣␣␣2182540253830879952893670507118343712989321426002991737403785142612646 9
␣␣␣␣5250897646270832307124860459187526487820547916176260627474993626505280 1
␣␣␣␣0110188010071218485412813152299550363 1439");
    BigInt res = num1 / sixteen;
    cout << "\n\nFirst\n" << sixteen << "\n";
    cout << "\n\nSecond\n" << res << "\n";
    cout << "\n\n";
    BigInt check = sixteen * res;
    cout << check << "\n";
    cout << num1 << "\n";
    if( check == num1 ){
        cout << "ok\n";
    }
    return 0;

}
```

Выводы

Факторизация целых чисел обеспечивается основной теоремой арифметики. По основной теореме арифметики каждое натуральное число имеет единственное разложение на простые множители. Существует множество алгоритмов факторизации целого, с помощью которых можно факторизовать любое натуральное число до состава его простых множителей с помощью рекуррентных формул. Однако, для очень больших чисел эффективный алгоритм пока неизвестен.
Факторизация больших чисел является задачей большой сложности. Не существует никакого известного способа, чтобы решить эту задачу быстро. После выполнения данной лабораторной работы я понял, что алгоритм шифрования RSA безопасен и на него можно пологаться в ближайшем будущем. По крайней мере пока квантовые компьютеры не доведут до совершенства.