Московский авиационный институт
(национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

# Лабораторная работа №1 по курсу «Криптография»

Студент: А. А. Литвина
Преподаватель: А. В. Борисов
Группа: М8О-306Б
Дата:
Оценка:
Подпись:

**Москва, 2020**

# Вариант №8

**Задача:**

Разложить каждое из чисел представленных ниже на нетривиальные сомножители.

**Первое число:**

12324826891193792319990614121664536366508704542268935810408918531614891149610З

**Второе число:**

1589686907858960532293041950259807409089116075774905924811928369 72
9308516275072914492447303882343619014782861146277474256634429223 57
3812679829885857722514236789773758073602382754296398746760528620 46
7135686904091857677298686613353160501421254539364215543462330529 17
3823254785957892596743971469331053694628704719897511634494080726 38
4449311911326430543608031846181210590808073104043168515626922519 39
36839179818736338280530681697503531374123421010923268140012860799 31

**Выходные данные:**

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

# 1 Описание

Для разложения числа на простые множители существует множество алгоритмов. Для первого числа я выбрала метод перебора возможных делителей. Это один из самых простых и очевидных алгоритмов факторизации, заключающийся в том, чтобы последовательно делить факторизуемое число n на натуральные числа от 1 до $\sqrt{n}$. Достаточно делить только на простые числа в этом интервале, для этого необходимо производить проверку каждого из чисел на простоту. Однако, как я поняла позднее, для очень больших чисел (таких, как мое) этот алгоритм будет работать тысячи лет. Кроме того, в моей реализации есть ограничения. Так как я писала программу на C++ и не использовала никаких дополнительных библиотек, деление чисел я реализовывала сама ("в столбик"), вследствии чего делитель (то есть каждый из множителей числа) ограничивается максимальным размером целочисленного типа данных. Такой вариант подойдет для небольших чисел или чисел, у которых много множителей. Поэтому я решила обратиться к другим методам решения задачи.

Для чисел меньше 100 знаков используется метод квадратичного решета. Этот метод считается одним из самых эффективных современных алгоритмов факторизации. Я решила воспользоваться готовой реализацией этого метода - программой msieve. Она справилась с поставленной задачей для 1-го числа менее чем за 1 минуту.

Однако 2-е число имеет более 400 знаков, факторизация которого за разумное время невозможна ни одним из ныне существующих алгоритмов. Но товарищи, которые уже выполнили эту лабораторную, рассказали мне, что один из множетелей этого числа - это наибольший общий делитель данного числа и числа из другого варианта. А второй множитель - это результат деления моего числа на первый множитель. Для работы с большими числами и поиска НОДа в этой программе я использовала библиотеку gmp.

## 2 Исходный код

Реализация алгоритма перебора возможных делителей на языке C++

```
#include <iostream>
#include <vector>
#include <limits.h>
#include <cmath>

using namespace std;

bool del(vector <int> v, int k) {
  int s, prev, i;
  s=0;
  prev=0;
  while (prev < v.size()) {
    for (i=prev; i<v.size(); i++) {
      s=s*10+v[i];
      if (s>=k)
        break;
    }
    prev=i+1;
    s=s%k;
  }

  if (s) return false;
  else return true;
}

bool simple(int k) {
  for (int i=2; i<=floor(sqrt(k)); i++) {
    if (k%i==0)
      return false;
  }
  return true;
}

int max_N(vector <int> v) {
  if (v.size()<10) {
    int R=0;
    for (int k=0; k<v.size(); k++) {
      R=R*10+v[k];
    }
    return floor(sqrt(R));
  }
  return INT_MAX;
}

int main() {
```

```
  vector <int> v =
     {1,2,3,2,4,8,2,6,8,9,1,1,9,3,7,9,2,3,1,9,9,9,0,6,1,4,1,2,1,6,6,4,5,3,6,3,6,6,5,0,8,7,0,4,5,4,2,

  vector <int> res;
  vector <int> answer;
  int j;

  for (j=2; j<=max_N(v); j++) {
    cout << j << "\t" << v.size() << endl;
    if ((del(v,j))&&(simple(j))) {
      answer.push_back(j);
      int s, prev, i;
      s=0;
      prev=0;
      while (prev < v.size()) {
        for (i=prev; i<v.size(); i++) {
          s=s*10+v[i];
          if (s>=j)
            break;

          if (i)
            res.push_back(0);
        }
        prev=i+1;
        res.push_back(s/j);
        s=s%j;
      }

      v=res;
      res.assign(0,0);
      j--;
    }
  }

  int R=0;
  for (int i=0; i<v.size(); i++) {
    R=R*10+v[i];
  }
  answer.push_back(R);

  for (int i=0; i<answer.size(); i++) {
    cout << answer[i] << "\t";
  }
  cout << endl;

}
```

Поиск НОДа для второго числа с помощью библиотеки gmp

```
for(int i=0; i<40; i++){
    mpz_gcd(nod.get_mpz_t(), v[i].get_mpz_t(), number.get_mpz_t());
    if((nod!=1) && (i!=n)){
        cout << "number1: " << nod << endl << endl;
        cout << "number2: " << number / nod << endl;
        break;
    }
}
```

# 3 Консоль

```
anast@anast-Lenovo-B590:~/Kripta$ msieve -m -q
next number: 123248268911937923199906141216645363665087045422689358104089185316148911496103

123248268911937923199906141216645363665087045422689358104089185316148911496103
p39: 321985376278994307302664413499387768503
p39: 382775982984847122295865568872934509201

anast@anast-Lenovo-B590:~/Kripta$ g++ 2.cpp -lgmpxx -lgmp
anast@anast-Lenovo-B590:~/Kripta$ ./a.out
number1: 1304738680325836098271854489803647581810257219
7915858405851096848067467531800758251299143837940990002
5634206602999335047253958597645751922572319519736549021
0983877340317419683868850874695648113695497565627917211
5606067161186055342248910459683542763941795236487891843
3203601720734649355543293580048106131166649

number2: 1218394864680461150604576510132382108126568195
7725881897412730838710933524329980502742744486796844773
634764902919516345655671446270292881515725992528646419
```

# 4 Ответ

**Разложение первого числа:**

- 321985376278994307302664413499387768503
- 382775982984847122295865568872934509201

**Разложение второго числа:**

- 1304738680325836098271854489803647581810257219
791585840585109684806746753180075825129914383794099000
256342066029993350472539585976457519225723195197365490
210983877340317419683868850874695648113695497565627917
211560606716118605534224891045968354276394179523648789
18433203601720734649355543293580048106131166649
- 1218394864680461150604576510132382108126568195
7725881897412730838710933524329980502742744486796844773
634764902919516345655671446270292881515725992528646419

# 5 Выводы

В данной лабораторной работе я познакомилась с факторизацией больших чисел. Эта работа научила меня тому, что прежде чем приступать к выполнению задачи тем или иным методом, необходимо подумать, сколько ресурсов она потребует. Так, метод простого перебора, который превым пришел мне на ум, оказался, к сожалению, абсолютно бесполезен, ведь он будет выполнять данную задачу на моем компьютере тысячи лет. Даже метод решета - самый эффективный метод факторизации на данный момент, хоть и лучше справляется с данной задачей, но также оказывается бессилен для чисел с количеством знаков больше 100.

Поняв, насколько, оказывается, сложно факторизовать большое число, я оценила значимость метода шифрования под названием RSA.

# Список литературы


*Факторизация целых чисел*
URL: https://ru.wikipedia.org/wiki/Факторизация_целых_чисел

*Перебор делителей*
URL: https://ru.wikipedia.org/wiki/Перебор_делителей

*Общий метод решета числового поля*
URL: https://ru.wikipedia.org/wiki/Общий_метод_решета_числового_поля

*Метод квадратичного решета*
URL: https://ru.wikipedia.org/wiki/Метод_квадратичного_решета

*Библиотека GMP*
URL: https://gmplib.org/