Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра информационных](https://www.belstu.by/fakultety/fit/vm) систем и технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №3**

по дисциплине Информационная безопасность

Тема: Основы теории чисел и их использование в криптографии

Студент: Станчик М.А.

ФИТ курс 3 группа 4

Преподаватель: Савельева М.Г.

Минск, 2025

**Лабораторная работа №3**

**Цель:** приобретение практических навыков выполнения операций с числами для решения задач в области криптографии и разработка приложений для автоматизации этих операций.

**Задачи:**

1. Закрепить теоретические знания по высшей арифметике.

2. Научиться практически решать задачи с использованием простых и взаимно простых чисел, вычислений по правилам модулярной арифметики и нахождению обратных чисел по модулю.

3. Ознакомиться с особенностями реализации готового программного средства L\_PROST и особенностями выполнения с его помощью операций над простыми числами.

4. Разработать приложение для реализации указанных преподавателем операций с числами.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения эксперимента с использованием приложения и результатов эксперимент.

# Теоретические сведения

Множество всех целых чисел (обозначим буквой *Z*) есть набор всех действительных чисел без дробной части: {..., –3, –2, –1, 0, 1, 2, 3, ...}.

Натуральные числа являются подмножеством целых чисел и образуют множество *N*: {1, 2, 3, ...}.

Делимость – одно из основных понятий теории чисел. Если для некоторого целого числа a и натурального числа *b* существует целое число *q*, при котором *bq* = *a*, то говорят, что число a делится на *b*. В этом случае *b* называется делителем числа a, а a называется кратным числу *b*.

Делитель *a* называется собственным делителем числа *b*, если 1 < *|a|* < *|b|*, и несобственным – в противном случае.

Всякое целое число, а можно представить с помощью положительного целого числа b равенством вида, *а* = *bq* + *r*, 0 ≤ *r* ≤ *b*. Число *q* называется неполным частным, а число *r* – остатком отделения, *а* на *b*.

Натуральное число n называется простым, если *n* > 1 и не имеет положительных делителей, отличных от 1 и *n*.

Любое составное число представляется уникальным образом в виде произведения простых чисел; иначе еще говорят, что разложение числа на простые множители однозначно.

Всякое натуральное число *n*, кроме 1, можно представить, как произведение простых множителей: *n* = *p1p2p3*...*pz*, *z* > 1.

Простых чисел бесконечно много, причем существует примерно *n* / ln(*n*) простых чисел, меньших числа *n*.

Наименьший простой делитель составного числа n не превышает √*n*, поэтому для проверки простоты числа достаточно проверить его делимость на 2 и на все нечетные (а еще лучше простые) числа, не превосходящие √*n*.

Любое четное число, большее 2, представимо в виде суммы двух простых чисел, а любое нечетное, большее 5, представимо в виде суммы трех простых чисел.

Для любого натурального *n*, большего 1, существует хотя бы одно простое число на интервале от *n* до 2*n*.

Натуральное число *n* называется составным, если *n* > 1 и имеет по крайней мере один положительный делитель, отличный от 1 и *n*.

Положительный наименьший собственный делитель составного числа *n* есть простое число.

Если два простых числа отличаются на 2, то их называют числами-близнецами.

Первый алгоритм нахождения простых чисел, не превышающих n, был придуман Эратосфеном во II в. до н. э. и известен сейчас как «решето Эратосфена». Его суть в последовательном исключении из списка целых чисел от 1 до *n* (или из сокращенного диапазона, например от m до n, 1 < *m* ≤ *n*) чисел, кратных 2, 3, 5 и другим простым числам, уже найденным «решетом».

Наибольшее целое число, которое делит без остатка числа *a* и *b*, называется наибольшим общим делителем этих чисел – НОД (*a*, *b*). Простым и эффективным средством вычисления НОД (*a*, *b*) является алгоритм Евклида.

Взаимно простыми являются целые числа, наибольший общий делитель которых равен 1.

Целые числа *a* и *b* взаимно просты тогда и только тогда, когда существуют такие целые числа *u* и *v*, что выполняется равенство:

*аu* + *bv* = 1.

Если НОД (*a*, *b*) = *d*, то справедливо следующее соотношение (соотношение Безу):

*аu* + *bv* = *d*.

Количество натуральных чисел, не превосходящих *n* и взаимно простых с *n*, называется функцией Эйлера и обозначается φ(*n*).

Если *p* – простое число, то φ(*p*) = *p* – 1, если числа *p* и *q* являются простыми и *p* ≠ *q*, то φ(*p*) = (*p* – 1)(*q* – 1).

Модулярная арифметика так же коммутативна, ассоциативна и дистрибутивна, как и обычная арифметика. В силу этих свойств сравнения можно почленно складывать, вычитать, умножать, возводить в степень.

Малая теорема Ферма. Если *n* – простое число, а число *а* не кратно *n*, то справедливо:

*an* ≡ 1 mod *n*.

В соответствии с обобщением Эйлера приведенной теоремы, если НОД (*а*, *n*) = 1, то справедливо:

*a*φ(*n*) mod *n* ≡ 1

Последнее выражение можно переписать в следующем виде:

*а*–1 mod *n* ≡ *a*φ(*n*) – 1 mod *n*.

# Ход работы

Целью работы была разработка приложения, позволяющего вычислять НОД двух или трех чисел и выполнять поиска простых чисел.

Для вычисления НОД был разработан метод gcd, принимающая три числа (последнее число является необязательным параметром). Этот подход основан на алгоритме Евклида, который гласит, что НОД двух чисел также равен НОД второго числа и остатка от деления первого числа на второе. Код метода представлен в листинге 2.1.

public double gcd(double firstNumber, double secondNumber, double thirdNumber) {

firstNumber = Math.abs(firstNumber);

secondNumber = Math.abs(secondNumber);

if (thirdNumber == 0) {

return gcd(firstNumber, secondNumber);

} else {

thirdNumber = Math.abs(thirdNumber);

double gcdAB = gcd(firstNumber, secondNumber);

return gcd(gcdAB, thirdNumber);

}

}

public double gcd(double firstNumber, double secondNumber) {

firstNumber = Math.abs(firstNumber);

secondNumber = Math.abs(secondNumber);

return secondNumber == 0 ? firstNumber : gcd(secondNumber, firstNumber % secondNumber);

}

Листинг 2.1 – Метод для подсчета НОД по алгоритму Евклида

Результат работы метода представлен на рисунке 2.1.

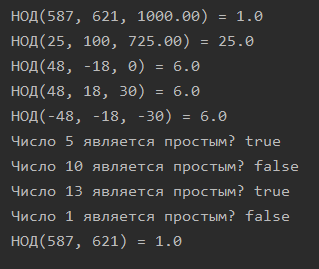


Рисунок 2.1 – Результат подсчета НОД для двух и трех чисел

Также был разработан метод isPrime, которая принимает число и проверяет, является ли оно простым. Код метода представлен в листинге 2.2.

public boolean isPrime(int a) {

if (a < 2) {

return false;

}

else {

int square = (int) Math.round(Math.sqrt(a));

for (int i = 2; i <= square; i++) {

if (a % i == 0) {

return false;

}

}

return true;

}

}

Листинг 2.2 – Метод для определения простоты числа

С помощью метода isPrime было определено, является ли конкатенация чисел *m* и *n* простым числом. Результат работы функции представлен на рисунке 2.2.

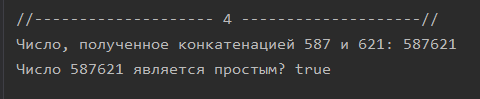


Рисунок 2.2 – Результат работы функции isPrime

Также была разработан метод findPrimeFactors, который позволяет записать число в виде произведения простых множителей в канонической форме. Код метода представлен в листинге 2.3.

public static List<Integer> findPrimeFactors(int n) {

List<Integer> factors = new ArrayList<>();

int divisor = 2;

while (n > 1) {

if (n % divisor == 0) {

factors.add(divisor);

n /= divisor;

} else { divisor++;

}

}

return factors; }

Листинг 2.3 – Код метода findPrimeFactors

Также была разработан метод formatPrimeFactors для форматирования списка простых множителей числа в строку, которая отображает их в удобочитаемом виде с учетом степеней. Код метода представлен в листинге 2.4.

public static String formatPrimeFactors(List<Integer> factors) {

Map<Integer, Integer> factorMap = new HashMap<>();

for (int factor : factors) {

factorMap.put(factor, factorMap.getOrDefault(factor, 0) + 1);

}

StringBuilder result = new StringBuilder();

for (Map.Entry<Integer, Integer> entry : factorMap.entrySet()) {

if (result.length() > 0) {

result.append(" \* ");

}

result.append(entry.getKey());

if (entry.getValue() > 1) {

result.append("^").append(entry.getValue());

}

}

return result.toString();

}

Листинг 2.4 – Код метода findPrimeFactors

Результат работы метода представлен на рисунке 2.3.

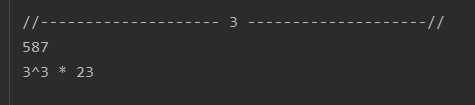


Рисунок 2.3 – Результат работы методов findPrimeFactors и formatPrimeFactors

Далее был разработан метод sieveOfEratosthenes, которая по алгоритму «решето Эратосфена» находит все простые числа в интервале от *m* до *n*. Метод возвращает список простых чисел, найденных в интервале от *m* до *n* (включительно). Код представлен в листинге 2.5.

public List<Integer> sieveOfEratosthenes(int m, int n) {

boolean[] isPrime = new boolean[n + 1];

for (int i = 0; i <= n; i++) {

isPrime[i] = true;

}

for (int i = 2; i \* i <= n; i++) {

if (isPrime[i]) {

for (int j = i \* i; j <= n; j += i) {

isPrime[j] = false;

}

}

}

List<Integer> primes = new ArrayList<>();

for (int i = Math.max(2, m); i <= n; i++) {

if (isPrime[i]) {

primes.add(i);

}

}

return primes;

}

Листинг 2.5 – Метод для нахождения простых чисел в заданном интервале

Результат работы метода представлен на рисунке 2.4.

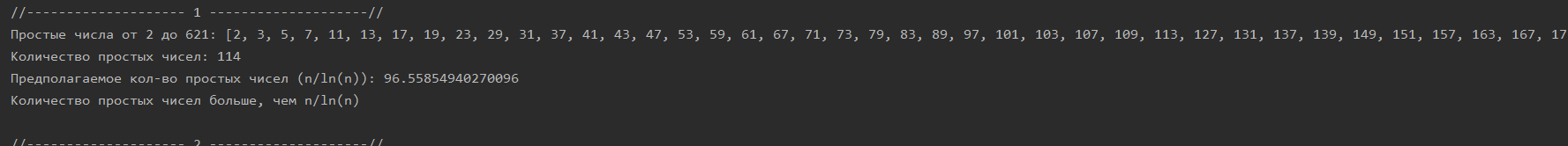


Рисунок 2.4 – Результат работы метода для нахождения простых чисел для *m* = 587, *n* = 621

Результат работы метода нужно было сравнить с ручными вычислениями, используя алгоритм «решето Эратосфена».

Порядок выполнения:

1. Исходный массив чисел: 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621.
2. Вычисление √621 ≈ 24,92. Значит, для проверки простоты числа, достаточно проверить его делимость на 2 и на все простые числа, не превосходящие √*n*. Значит проверять делимость нужно будет на следующие числа: 2, 3, 5, 7, 11, 13, 17, 19, 23.
3. Удаление чисел из массива с учётом *s* = 2: 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607, 609, 611, 613, 615, 617, 619, 621.
4. Удаление чисел из массива с учётом *s* = 3: 587, 589, 593, 595, 599, 601, 605, 607, 611, 613, 617, 619, 621.
5. Удаление чисел из массива с учётом *s* = 5: 587, 589, 593, 599, 601, 607, 611, 613, 617, 619, 621.
6. Удаление чисел из массива с учётом *s* = 7: 587, 589, 593, 599, 601, 607, 611, 613, 617, 619, 621.
7. Удаление чисел из массива с учётом *s* = 11: 587, 589, 593, 599, 601, 607, 611, 613, 617, 619, 621.
8. Удаление чисел из массива с учётом *s* = 13: 587, 589, 593, 599, 601, 607, 613, 617, 619, 621.
9. Удаление чисел из массива с учётом *s* = 17: 587, 589, 593, 599, 601, 607, 613, 617, 619, 621.
10. Удаление чисел из массива с учётом *s* = 19: 587, 593, 599, 601, 607, 613, 617, 619, 621.
11. Удаление чисел из массива с учётом *s* = 23: 587, 593, 599, 601, 607, 613, 617, 619.

Итоговый массив: [587, 593, 599, 601, 607, 613, 617, 619], что совпадает с массивом чисел, полученном при работе приложения.

**Вывод:** в ходе выполнения лабораторной работы были освоены основы теории чисел, необходимые для применения в криптографии. Это включало в себя изучение свойств простых и составных чисел, взаимной простоты чисел, а также критериев делимости. Целью работы было укрепление теоретических знаний в области высшей арифметики и приобретение практических навыков выполнения операций с числами, включая операции модулярной арифметики и нахождение обратных чисел по модулю. В результате работы было разработано приложение, позволяющее автоматизировать такие операции, как вычисление НОД и нахождение простых чисел, что способствует эффективному решению задач в области криптографии.