Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надежность информационных систем»

**Отчёт по лабораторной работе №12**

Основы теории чисел и их использование в криптографии

Студент: Жук С.С.

ФИТ 3 курс 2 группа

Преподаватель: Савельева М.Г.

**Содержание**

[1 Теоретические сведения 3](#_Toc196675067)

[2 Нахождение наибольшего общего делителя 7](#_Toc196675068)

[3 Нахождение простых чисел 8](#_Toc196675069)

[4 Произведение простых множителей в канонической форме 11](#_Toc196675070)

[Вывод 12](#_Toc196675071)

# **1 Теоретические сведения**

Мы неоднократно подчеркивали, что каждый из естественных языков обладает избыточностью. Среди европейских языков белорусский и русский обладают одним из самых высоких уровней избыточности.

В основе современной криптографии лежит теория чисел. Теория чисел – раздел математики, изучающий натуральные числа и иные похожие величины.

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

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

Определение 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 – делитель числа *a*, а *a* – кратное числу *b*. При этом используются следующие обозначения:

*a* ⋮ *b* – *a* делится на *b*, или *b* | *a* – *b* делит *a*.

Из последнего определения следует, что:

* любое натуральное число является делителем нуля;
* единица является делителем любого целого числа;
* любое натуральное число является делителем самого себя.

Определение 4. Делитель *a* – собственный делитель числа *b*, если 1 < |*a*| < |*b*|, и несобственным – в противном случае.

Определение 5. 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*.

Каждое натуральное число, большее единицы, делится по крайней мере на два числа: на 1 и на само себя.

Если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным.

Определение 6. Натуральное число *n* называется простым, если *n* > 1 и не имеет положительных делителей, отличных от 1 и *n*.

Простое число не делится без остатка ни на одно другое число.

Перечислим несколько важных свойств простых чисел.

Свойство 1. Любое составное число представляется уникальным образом в виде произведения простых чисел; иначе еще говорят, что разложение числа на простые множители однозначно.

Основная теорема арифметики. Всякое натуральное число *n*, кроме 1, можно представить как произведение простых множителей.

Каноническая форма – порядок записи сомножителей в порядке возрастания.

Свойство 2. Простых чисел бесконечно много, причем существует примерно *n* / ln(*n*) простых чисел, меньших числа *n*.

Свойство 3. 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*.

Из соотношения *n* = *qp* натуральных чисел, больших единицы, следует, что либо *p*, либо *q* принадлежит отрезку от 2 до √*n*.

Поиск сомножителей числа *n* может вестись, например, перебором всех простых чисел до √*n*. Однако если множители – большие простые числа, то на их поиск может потребоваться много времени.

Сложность решения задачи разложения больших чисел на простые сомножители, известной как «проблема факторизации», определяет криптостойкость некоторых алгоритмов асимметричной криптографии, в частности алгоритма RSA.

Свойство 4. Любое четное число, большее 2, представимо в виде суммы двух простых чисел, а любое нечетное, большее 5, представимо в виде суммы трех простых чисел.

Свойство 5. Для любого натурального n, большего 1, существует хотя бы одно простое число на интервале от *n* до 2*n*.

Определение 7. Натуральное число *n* – составное, если *n* > 1 и имеет по крайней мере один положительный делитель, отличный от 1 и *n*.

Единица не считается ни простым числом, ни составным.

Свойство 2 собственного делителя. Положительный наименьший собственный делитель составного числа *n* есть простое число.

Так как простое число не делится ни на какое другое, кроме себя самого, очевидный способ проверки числа *n* на простоту – разделить *n* на все числа (*n* – 1) и проанализировать наличие остатка от деления. Однако перебор может оказаться достаточно трудоемким, если на простоту нужно проверить число с количеством цифр в несколько десятков.

Существуют правила, способные заметно сократить время вычислений.

Правило 1. Воспользоваться свойством 3 простых чисел.

Правило 2. Если последняя цифра анализируемого числа является четной, то это число заведомо составное.

Правило 3. Числа, делящиеся на 5, всегда оканчиваются пятеркой или нулем. Если младшим разрядом анализируемого числа являются 5 или 0, то такое число не является простым.

Правило 4. Если анализируемое число делится на 3, то и сумма его цифр тоже обязательно делится на 3.

Правило 5. Основано на свойстве делимости на 11. Нужно из суммы всех нечетных цифр числа вычесть сумму всех четных его цифр. Четность и нечетность определяется счетом от младшего разряда. Если получившаяся разность делится на 11, то и анализируемое число тоже на него делится.

Правило 6. Основано на свойстве делимости на 7 и 13. Нужно разбить анализируемое число на тройки цифр, начиная с младших разрядов. Просуммировать числа, стоящие на нечетных позициях, и вычесть из них сумму чисел на четных. Проверить делимость результата на числа 7 и 13.

Определение 8. Если два простых числа отличаются на 2, то их называют числами-близнецами. Таких чисел не очень много. Например, ими являются 5 и 7, 29 и 31, 149 и 151.

Для нахождения всех простых чисел не больше заданного числа *n* в соответствии с «решетом Эратосфена» нужно выполнить следующие шаги:

1. выписать подряд все целые числа от двух (либо от *m*) до *n* (2, 3, 4, …, *n*). Пусть некоторая переменная (например, *s*) изначально равна 2 – первому простому числу;
2. удалить из списка числа от 2*s* до *n*, считая шагами по *s* (это будут числа, кратные s: 2*s*, 3*s*, 4*s*, …);
3. найти первое из оставшихся чисел в списке, большее чем *s*, и присвоить значению переменной *s* это число;
4. повторять шаги 2 и 3, пока возможно.

Определение 9. 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. В основе алгоритма лежит определение 5. В соответствии с этим определением используется цепочка вычислений двумя исходными (начальными) числами *а* и *b*.

Чтобы найти НОД нескольких чисел (например, *a*, *b*, *c*), достаточно найти НОД двух чисел (например, НОД (*a*, *b*) = *d*), потом НОД полученного (НОД (*a*, b)) и следующего числа (НОД (*c*, *d*)), и т. д.

Таким образом, чтобы вычислить НОД *k* чисел, нужно последовательно вычислить (*k* – 1) НОД. Последнее вычисление дает искомый результат.

Определение 10. Взаимно простыми являются целые числа, наибольший общий делитель которых равен 1.

Теорема 1. Целые числа *a* и *b* взаимно просты тогда и только тогда, когда существуют такие целые числа *u* и *v*, что выполняется равенство

(1.1)

Теорема 2. Если НОД (*a*, *b*) = *d*, то справедливо следующее соотношение (соотношение Безу):

(1.2)

Формула (1.2) называется также реализацией «расширенного алгоритма Евклида». Этот алгоритм состоит из двух этапов: собственно алгоритма Евклида и вычислений на основе обратных подстановок или последовательного выражения остатков в каждом из шагов предыдущего этапа с соответствующим приведением подобных на каждом шаге.

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

(1.3)

В модульной арифметике интересуемся остатком от деления числа *а* на число *n* (*n* – натуральное число и *n* > 1). Если таким остатком является число *b*, то можно записать:

*a* ≡ *b* (mod *n*), или *a* ≡ *b* mod *n*.

Такая формальная запись читается как «*a* сравнимо с *b* по модулю *n*».

При целочисленном (в том числе и нулевом) результате деления числа *а* на число *n* справедливо: *a* = *b* + *kn*.

Пример 19. Справедливы следующие сравнения чисел: –5 ≡ 7 mod 4 ≡ 11 mod 4 ≡ 23 mod 4 ≡ 3 mod 4.

Иногда *b* называют вычетом по модулю *n*, а числа *a* и *b* называют сравнениями (по модулю n). 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 (*аm* mod *n* ≡ (*а* mod *n*)*m*, если *a* ≡ *b* mod *n*, то *am* ≡ *bm* (mod *n*)).

Обратные числа в модулярной арифметике. В модулярной арифметике запись уравнения в виде

(1.4)

предусматривает поиск таких значений *х* и *k*, которые удовлетворяют равенству

(1.5)

Общая задача решения уравнения (1.4) может быть сформулирована следующим образом: найти такое *х*, что

(1.6)

Уравнение (1.5) имеет единственное решение, если *а* и *n* – взаимно простые числа, в противном случае – решений нет.

Уравнение (1.5) можно переписать в ином виде:

(1.7)

Если НОД (*а*, *n*) = 1, то *а*-1*а* ≡ 1 mod *n*, где *а*-1 – число, обратное *а* по модулю *n*.

Справедливо также:

(1.8)

В силу приведенных рассуждений и обоснований выражению (1.8) удовлетворяют такие числа, при которых выполняется равенство

(1.9)

где *k* – целое число (результат деления *xy*/*n*).

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

(1.10)

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

(1.11)

Последнее выражение можно переписать в следующем виде:

(1.12)

# **2 Нахождение наибольшего общего делителя**

Опишем функцию, которая предназначена для вычисления наибольшего общего делителя (НОД) двух чисел *a* и *b* с использованием алгоритма Евклида. В начале проверяется, если *b* равно 0, то возвращается значение *a*, так как НОД числа и 0 равен самому числу. Если же *b* не равно 0, функция рекурсивно вызывает себя, передавая в качестве новых аргументов *b* и остаток от деления *a* на *b*. Этот процесс продолжается до тех пор, пока остаток не станет равным 0, и тогда возвращается значение НОД. Программная реализация функции показана в листинге 2.1.

|  |
| --- |
| const calculateGCD = (a, b) =>  b === 0 ? a : calculateGCD(b, a % b); |

Листинг 2.1 – Функция для вычисления НОД двух чисел

Следующим шагом будет описана функция, которая расширяет предыдущую функцию и позволяет вычислять НОД для трёх чисел. Вначале она вычисляет НОД для первых двух чисел, а затем вычисляет НОД полученного результата с третьим числом. Таким образом, она находит наибольший общий делитель для трёх чисел одновременно. Функция продемонстрирована в листинге 2.2.

|  |
| --- |
| const calculateGCD3 = (a, b, c) =>  calculateGCD(calculateGCD(a, b), c); |

Листинг 2.2 – Функция для вычисления НОД трех чисел

Результат показан на рисунке 2.1.

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.



Рисунок 2.1 – Вывод результата

# **3 Нахождение простых чисел**

Для начала опишем функцию, которая проверяет, является ли заданное число простым. Для чисел меньше или равных 1 функция сразу возвращает false, так как простое число должно быть больше 1. Если число меньше или равно 3, то оно автоматически считается простым (кроме 1). Далее, если число делится на 2 или 3, оно не может быть простым, и функция возвращает false. Для чисел, больших 3, функция проверяет возможные делители начиная с 5 и до квадратного корня из числа. Если найден делитель, то число не простое. Если делителей не найдено, функция возвращает true, подтверждая, что число простое. Код представлен в листинге 3.1.

|  |
| --- |
| function Prime(num) {  if (num <= 1) return false;  if (num <= 3) return true;  if (num % 2 === 0 || num % 3 === 0) return false;   for (let i = 5; i <= *Math*.sqrt(num); i++) {  if (num % i === 0) {  return false;  }  }  return true; } |

Листинг 3.1 – Функция проверки на простое число

Следующим шагом будет описана функция, которая находит все простые числа в интервале от *n* до *m*, включая границы. Для этого она перебирает все числа от *n* до *m* и для каждого вызывает функцию, чтобы проверить его на простоту. Все простые числа добавляются в массив. После завершения перебора функция возвращает объект, содержащий массив простых чисел и их количество. Функция продемонстрирована в листинге 3.2.

|  |
| --- |
| function findPrimes(n, m) {  const primes = [];   for (let i = n; i <= m; i++) {  if (Prime(i)) {  primes.push(i);  }  }   return {  primes,  count: primes.length  }; } |

Листинг 3.2 – Функция для нахождения простых чисел в интервале

Далее опишем функцию, которая реализует алгоритм «решето Эратосфена» для поиска всех простых чисел в интервале от *n* до *m*. Сначала создаётся массив, в котором все элементы инициализируются значением true, что обозначает, что все числа потенциально простые. Для чисел 0 и 1 значение ставится в false, так как они не являются простыми. Затем начинается основная часть алгоритма: для каждого числа начиная с 2, если оно помечено как простое, то все его кратные помечаются как составные. После завершения алгоритма все числа, оставшиеся помеченными как простые, добавляются в массив. Функция возвращает объект с массивом простых чисел и их количеством. Программная реализация функции показана в листинге 3.3.

|  |
| --- |
| function sieveOfEratosthenes(n1, m1) {  const num = *Array*(m1 + 1).fill(true);  num[0] = num[1] = false;   for (let i = 2; i <= *Math*.sqrt(m1); i++) {  if (num[i]) {  for (let j = i \* i; j <= m1; j += i) {  num[j] = false;  }  }  }   const primes = [];  for (let i = *Math*.max(n1, 2); i <= m1; i++) {  if (num[i]) {  primes.push(i);  }  }   return { primes, count: primes.length }; } |

Листинг 3.3 – Функция реализации алгоритма «решето Эратосфена»

Сравним полученные результаты с «ручными вычислениями», используя «решето Эратосфена»:

Шаг 1. Выпишем числа от 499 до 531: 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531.

Шаг 2. Удалим из списка числа с учетом *s* = 2: 499, 501, 503, 505, 507, 509, 511, 513, 515, 517, 519, 521, 523, 525, 527, 529, 531.

Шаг 3. Удалим из списка числа с учетом *s* = 3: 499, 503, 505, 509, 511, 517, 521, 523, 527, 529.

Шаг 4. Удалим из списка числа с учетом *s* = 5: 499, 503, 509, 511, 517, 521, 523, 527, 529.

Шаг 5. Удалим из списка числа с учетом *s* = 7: 499, 503, 509, 517, 521, 523, 527, 529.

Шаг 6. Удалим из списка числа с учетом *s* = 11: 499, 503, 509, 521, 523, 527, 529.

Шаг 7. Удалим из списка числа с учетом *s* = 13: 499, 503, 509, 521, 523, 527, 529.

Шаг 8. Удалим из списка числа с учетом *s* = 17: 499, 503, 509, 521, 523, 529.

Шаг 8. Удалим из списка числа с учетом *s* = 19: 499, 503, 509, 521, 523, 529.

Шаг 9. Удалим из списка числа с учетом *s* = 23: 499, 503, 509, 521, 523.

Таким образом, числа 499, 503, 509, 521, 523 являются простыми в диапазоне от 499 до 531. Как видим, количество таких чисел – 5.

Следующим шагом будет описана функция, которая проверяет, является ли число простым. Если число меньше 2, то оно не может быть простым. Для чисел, равных или больших 2, функция проверяет, делится ли число на числа от 2 до квадратного корня. Если находится делитель, то число не является простым. Если делители не найдены, число считается простым. Функция продемонстрирована в листинге 3.4.

|  |
| --- |
| const isPrime = (num) => {  if (num < 2) return { isPrime: false, divisor: null };  for (let i = 2; i <= *Math*.sqrt(num); i++) {  if (num % i === 0) {  return { isPrime: false, divisor: i };   }  }  return { isPrime: true, divisor: null };  }; |

Листинг 3.4 – Функция для проверки на простоту числа

Далее опишем функцию, которая проверяет, является ли число, полученное конкатенацией чисел *m* и *n*, простым. Для начала она вычисляет длину числа *n* в виде строки и использует её для вычисления конкатенированного числа. Конкатенация осуществляется путём умножения числа *m* на 10 в степени длины *n* и добавления числа *n*. После этого вызывается функция, чтобы проверить полученное число на простоту. Если оно простое, функция возвращает строку, подтверждающую, что конкатенированное число является простым. Если число не простое, возвращается строка, в которой указывается делитель, на который оно делится. Программная реализация функции показана в листинге 3.5.

|  |
| --- |
| const isConcatenatedNumberPrime = (m, n) => {  const nLength = n.toString().length;  const concatenatedNumber = m \* *Math*.pow(10, nLength) + n;  const result = isPrime(concatenatedNumber);  if (result.isPrime) {  return `Число, состоящее из конкатенации цифр ${m} и ${n}, является простым`;  } else {  return `Число, состоящее из конкатенации цифр ${m} и ${n}, не является простым и делится на ${result.divisor}`;  } }; |

Листинг 3.5 – Функция проверки на простое число

Результат показан на рисунке 3.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.



Рисунок 3.1 – Вывод результатов

# **4 Произведение простых множителей в канонической форме**

Для начала опишем функцию, которая находит все простые множители числа и возвращает их в виде массива. Она начинается с делителя, равного 2, и выполняет цикл, в котором делит число на текущий делитель, пока оно делится на него без остатка, добавляя делитель в массив множителей. После этого делитель увеличивается на 1, и процесс повторяется до тех пор, пока число не станет равным 1. Функция возвращает массив простых множителей числа. Код представлен в листинге 4.1.

|  |
| --- |
| const primeFactors = (num) => {  const factors = [];  let divisor = 2;  while (num > 1) {  while (num % divisor === 0) {  factors.push(divisor);  num /= divisor;  }  divisor++;  }  return factors; }; |

Листинг 4.1 – Функция нахождения всех простых множителей числа

Следующим шагом будет описана функция, которая принимает целое число и возвращает строку, представляющую его разложение на простые множители в каноническом виде. Она сначала получает простые множители числа с помощью функции, затем подсчитывает количество вхождений каждого множителя, и, если множитель встречается больше одного раза, добавляет его степень, в противном случае – просто сам множитель. Результатом работы функции является строка, представляющая число как произведение простых множителей, разделенных умножением. Функция продемонстрирована в листинге 4.2.

|  |
| --- |
| const canonForm = (num) => {  const factors = primeFactors(num);  const factorCounts = {};   for (const factor of factors) {  factorCounts[factor] = (factorCounts[factor] || 0) + 1;  }   const formattedFactors = *Object*.entries(factorCounts).map(([factor, count]) => {  return count > 1 ? `${factor}^${count}` : factor;  });   return `${num} = ${formattedFactors.join(' \* ')}`; }; |

Листинг 4.2 – Функция для получения простых множителей числа

Результат показан на рисунке 4.1.

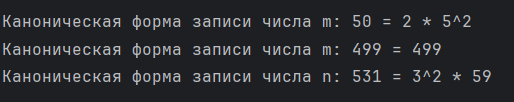


Рисунок 4.1 – Вывод результатов

# **Вывод**

В ходе выполнения лабораторной работы были приобретены практические навыки выполнения операций с числами, необходимые для решения задач в области криптографии. Также было разработано приложение, позволяющее автоматизировать данные операции, включая вычисление наибольшего общего делителя (НОД) для двух и более чисел, а также поиск простых чисел в заданных интервалах с использованием эффективных алгоритмов, таких как «решето Эратосфена». Эти навыки и приложение могут быть полезными при решении задач, связанных с криптографией, где необходимы быстрые и точные вычисления, включая работу с простыми числами и делителями.