## Front matter

title: “Отчет по Лабораторной работе №5 по предмету Математические основы защиты информации и кибер безопасности” author: “Лобов Михаил Сергеевич”

## Generic otions

lang: ru-RU toc-title: “Содержание”

## Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

## Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt ## I18n polyglossia polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english ## I18n babel babel-lang: russian babel-otherlangs: english ## Fonts mainfont: IBM Plex Serif romanfont: IBM Plex Serif sansfont: IBM Plex Sans monofont: IBM Plex Mono mathfont: STIX Two Math mainfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 romanfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 sansfontoptions: Ligatures=Common,Ligatures=TeX,Scale=MatchLowercase,Scale=0.94 monofontoptions: Scale=MatchLowercase,Scale=0.94,FakeStretch=0.9 mathfontoptions: ## Biblatex biblatex: true biblio-style: “gost-numeric” biblatexoptions: - parentracker=true - backend=biber - hyperref=auto - language=auto - autolang=other\* - citestyle=gost-numeric ## Pandoc-crossref LaTeX customization figureTitle: “Рис.” tableTitle: “Таблица” listingTitle: “Листинг” lofTitle: “Список иллюстраций” lotTitle: “Список таблиц” lolTitle: “Листинги” ## Misc options indent: true header-includes: -

# keep figures where there are in the text

## # keep figures where there are in the text

# Цель работы

Изучить “Вероятностные алгоритмы проверки чисел на простоту”.

# Задание

Реализовать алгоритмы проверки чисел на простоту на языке Julia.

# Теоретическое введение

Пусть – целое число. Числа называются тривиальными делителями числа .

Целое число называется простым, если оно не является делителем единицы и не имеет других делителей, кроме тривиальных. В противном случае число называется составным.

Например, числа являются простыми.

Пусть . Целые числа и называются сравнимыми по модулю (обозначается ) если разность делится на . Также эта процедура называется нахождением остатка от целочисленного деления на .

Проверка чисел на простоту является составной частью алгоритмов генерации простых чисел, применяемых в криптографии с открытым ключом. Алгоритмы проверки на простоту можно разделить на вероятностные и детерминированные.

Детерминированный алгоритм всегда действует по одной и той же схеме и гарантированно решает поставленную задачу (или не дает никакого ответа). Вероятностный алгоритм использует генератор случайных чисел и дает не гарантированно точный ответ. Вероятностные алгоритмы в общем случае не менее эффективны, чем детерминированные (если используемый генератор случайных чисел всегда дает набор одних и тех же чисел, зависящих от входных данных, то вероятностный алгоритм становится детерминированным).

Для проверки на простоту числа вероятностным алгоритмом выбирают случайное число , такое что , и проверяют условия алгоритма. Если число не проходит тест по основанию , то алгоритм выдает результат «Число составное», и число действительно является составным.

Если же проходит тест по основанию , ничего нельзя сказать о том, действительно ли число является простым. Последовательно проведя ряд проверок таким тестом для разных и получив для каждого из них ответ «Число , вероятно, простое», можно утверждать, что число является простым с вероятностью, близкой к 1. После независимых выполнений теста вероятность того, что составное число будет раз объявлено простым (вероятность ошибки), не превосходит .

# Выполнение лабораторной работы

## Написаны программы на языке Julia.

### Тест Ферма

Тест Ферма основан на малой теореме Ферма: для простого числа и произвольного числа , такого что , выполняется сравнение

Следовательно, если для нечетного существует такое целое , что и , то число составное. Отсюда получаем следующий вероятностный алгоритм проверки числа на простоту.

using Random  
n = 20  
function is\_prime(n::Int,k::Int=5)  
 if n < 5 || n%2 == 0  
 return false  
 end  
 for \_ in 1:k  
 a = rand(2:n-2)   
 r = powermod(a,n-1,n)  
 if r != 1  
 return false  
 end  
 end  
 return true  
end  
if is\_prime(n)  
 println("Число $n скорее всего простое")  
else  
 println("Число $n составное")  
end

Число 20 составное

## 1. Пояснение програмного кода

**Вход:** Нечетное целое число .

**Выход:** «Число , вероятно, простое» или «Число составное».

1. Выбрать случайное целое число , такое что .
2. Вычислить .
3. При , результат: «Число , вероятно, простое». В противном случае результат: «Число составное».

## Вычисление символа Якоби

Необходим для **теста Соловея-Штрассена**

**Вход:** Нечетное целое число , целое число , такое что .

**Выход:** Символ Якоби .

1. Положить .
2. Если , результат: 0.
3. Если , результат: .
4. Представить в виде , где число нечетное.
5. При четном , положить ; при нечетном , положить , если ; положить , если .
6. Если , результат: .
7. Если и , то .
8. Положить и вернуться на шаг 2.

function jacobi\_symbol(a::Int, n::Int)  
 g = 1  
 if a == 0  
 return 0  
 end  
 if a == 1  
 return g  
 end  
 while a != 0  
 k = 0  
 while a % 2 == 0  
 a ÷= 2  
 k += 1  
 end  
 a1 = a  
 if k % 2 == 0  
 s = 1  
 else  
 if n % 8 == 1 || n % 8 == 7  
 s = 1  
 elseif n % 8 == 3 || n % 8 == 5  
 s = -1  
 end  
 end  
 g \*= s  
 if a1 == 1  
 return g  
 end  
 if n % 4 == 3 && a1 % 4 == 3  
 g = -g  
 end  
 a, n = n % a1, a1  
 end  
 return g  
end  
  
n = 19  
a = rand(0:n-2)  
result = jacobi\_symbol(a, n)  
println("Символ Якоби ($a/$n) = $result")

Символ Якоби (16/19) = 1

## Тест Соловея-Штрассена

Тест Соловея–Штрассена основан на критерии Эйлера: нечетное число является простым тогда и только тогда, когда для любого целого числа , такого что и взаимно простого с , выполняется сравнение:

где – символ Якоби.

Пусть , где и числа простые (не обязательно различные). Символ Якоби определяется равенством

n = 27  
  
function test\_solovei\_strassen(n::Int)  
 a = rand(2:n-2)  
 n\_1 = (n-1)/2  
 r = powermod(a, n\_1, n) # power(a, (n-1)/2, n)  
 if r != 1 && r !=(n-1)  
 return false  
 end  
 s = jacobi\_symbol(a,n)  
 if r == s%n  
 return true  
 else  
 return false  
 end  
end  
  
if is\_prime(n)  
 println("Число $n скорее всего простое")  
else  
 println("Число $n не является простым")  
end

Число 19 скорее всего простое

## Алгоритм, реализующий тест Миллера–Рабина

#### Описание алгоритма:

**Вход:** Нечетное целое число .

**Выход:** «Число , вероятно, простое» или «Число составное».

1. Представить в виде , где число нечетное.
2. Выбрать случайное целое число , такое что .
3. Вычислить .
4. При и , выполнить следующие действия:
   1. Положить .
   2. Если и , то
      1. Положить .
      2. Если , результат: «Число составное».
      3. Положить .
   3. При , результат: «Число составное».
5. Результат: «Число , вероятно, простое».

n = 19  
  
function test\_miller\_rabin(n::Int, k::Int=5)  
 s = 0  
 r = n - 1  
 while r % 2 == 0  
 r ÷= 2  
 s += 1  
 end  
  
 for \_ in 1:k  
 a = rand(2:n-2)  
 y = powermod(a, r, n)  
  
 if y == 1 || y == n - 1  
 continue  
 end  
  
 for j in 1:s-1  
 y = powermod(y, 2, n)  
 if y == 1  
 return false  
 end  
 if y == n - 1  
 break  
 end  
 end  
  
 if y != n - 1  
 return false  
 end  
 end  
 return true  
end  
  
if test\_miller\_rabin(n)  
 println("Число $n вероятно простое")  
else  
 println("Число $n составное")  
end

Число 19 вероятно простое

# Выводы

В ходе выполнения лабораторной работы №5 были изучены и реализованы на языке Julia вероятностные алгоритмы проверки чисел на простоту, такие как тест Ферма, тест Соловея — Штрассена и тест Миллера — Рабина. Эти алгоритмы играют важную роль в криптографии и других областях, где необходимо быстро определять простоту чисел. Вероятностные алгоритмы позволяют получать ответ с высокой степенью вероятности, однако, в отличие от детерминированных методов, не гарантируют абсолютной точности. Такой подход позволяет находить простые числа более эффективно, что особенно актуально при работе с большими числами. Алгоритм Миллера-Рабина является наиболее используемым в современных системах, потому что дает наибольшую точность.

# Список литературы