Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Проверка чисел на простоту**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

Лабораторная работа №2

студентки 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Громовой Наталии Викторовны

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2018

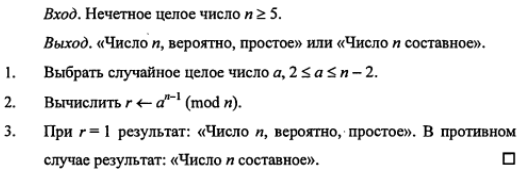
**Цель работы**: изучение основных методов проверки простоты чисел и их программная реализация.

**Теоретическая часть**

1. Тест Ферма

Согласно *малой теореме Ферма* если 𝑝 – простое число и 𝑎 – целое число, не делящееся на 𝑝, то 𝑎p-1 делится на 𝑝, то есть 𝑎p-1 ≡ 1 (𝑚𝑜𝑑 𝑝).Следовательно, если для нечетного *n* существует такое целое *a*, что , НОД (*а*,*n*)=1 и 𝑎n-1 ≡ 1 (𝑚𝑜𝑑 n), то число *n*  вероятно простое.

Алгоритм вычисления:



Сложность теста Ферма равна *О*(log3*n*). Тест имеет существенный недостаток в виде наличия чисел Кармайкла. Это нечетные составные числа, для которых сравнение из формулы выполняется при любом *a*,  , взаимно простом с *n*. Для всех *a*, НОД (*а*,*n*)=1, тест будет выдавать ошибочный результат.

b. Тест Соловея – Штрассена

Вероятностный тест простоты, распознает числа Кармайкла как составные, параметризуется количеством раундов 𝑘. В каждом раунде случайным образом выбирается число 𝑎 < 𝑛.

В основе этого теста лежит следующая теорема **(***критерий Эйлера***)**: нечетное число *n* является простым тогда и только тогда, когда для любого целого числа *a*, , взаимно простого с *n*, выполняется сравнение:

*,*

где - символ Якоби от параметров *a* и 𝑛.

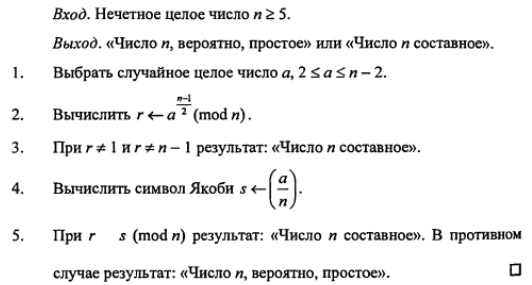
Если НОД(𝑎, 𝑛) > 1, то 𝑛 составное. Иначе проверяется справедливость сравнения *.*Если оно не выполняется, то 𝑛 — составное. Если это сравнение выполняется, то *a* является свидетелем простоты числа *n*.

Далее выбирается другое случайное *a* и процедура повторяется. После

нахождения *k* свидетелей простоты в 𝑘 раундах выносится заключение, что 𝑛

является простым числом с вероятностью 1 – 2-k.

Алгоритм вычисления:



Сложность теста Соловэя-Штрассена составляет *О*(log3*n*) из-за вычисления символа Якоби.

1. Тест Рабина – Миллера

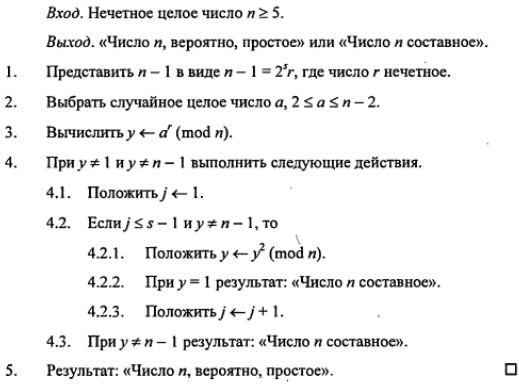
Определим свидетелей простоты числа *р*. Пусть 𝑝 > 2 – простое число. Представим число 𝑝 − 1 в виде 𝑝 − 1 = 2s𝑑, где 𝑑 – нечетно. Тогда число 𝑎 из назовем свидетелем простоты числа *р,* если выполняются:

𝑎*d* ≡ 1 (𝑚𝑜𝑑 𝑝)

∃𝑟, 0 ≤ 𝑟 ≤ 𝑠 − 1 ∶ ≡ −1(𝑚𝑜𝑑 𝑝).

Если *k* случайно выбранных чисел окажутся свидетелями простоты числа *р*, то вероятность, что *р* составное не превышает 4-*k*.

Алгоритм вычисления:



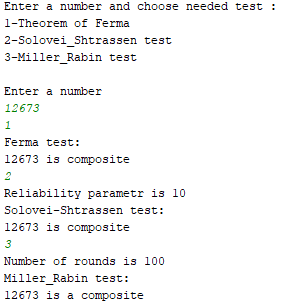
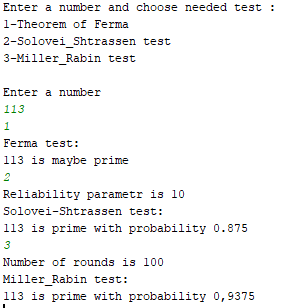
Сложность вычисления алгоритма *О*((log*n*)3).

**Практическая часть**

Все вышеописанные алгоритмы были написаны на языке Java. Листинги кода предоставлены в приложении.

Пример работы (анализ корректности)

Число 113 Число 12673



При использовании вероятностного характера тестов возможно ошибочное определение составных чисел. Исследуем зависимость ошибок от числа повторов.

Проверка ошибок для простого числа 6653.Количество прогонов – 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число повторов | Тест Ферма | Тест Соловэя-Штрассена | Тест Миллера-Рабина |
| 1 | 213 | 203 | 36 |
| 2 | 56 | 87 | 9 |
| 3 | 17 | 10 | 2 |
| 4 | 1 | 0 | 0 |

Проверка ошибок для числа 1729 (является числом Кармайкла). Количество прогонов – 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число повторов | Тест Ферма | Тест Соловэя-Штрассена | Тест Миллера-Рабина |
| 1 | 1000 | 325 | 104 |
| 2 | 1000 | 151 | 3 |
| 3 | 1000 | 66 | 0 |
| 4 | 1000 | 30 | 0 |

Число ошибок, допущенных в тесте Ферма, в зависимости от вида числа. Количество прогонов – 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число повторов | Число ошибок | | |
| n=645 | n=2147 | n=1365 |
| 1 | 567 | 93 | 15 |
| 2 | 193 | 20 | 1 |
| 3 | 104 | 5 | 0 |

Число ошибок, допущенных в тесте Миллера-Рабина, в зависимости от вида числа.

Количество прогонов – 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число повторов | Число ошибок | | |
| 645 | 2147 | 1365 |
| 1 | 15 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |

Число 1365 является сильно составным и в его состав не входят большие простые множители, в связи с этим, результаты для этого числа получились лучше.

Тестирование на время (c) (вводимое число 3541-простое):

|  |  |  |  |
| --- | --- | --- | --- |
| Число повторов | Тест Ферма | Тест Соловэя-Штрассена | Тест Миллера-Рабина |
| 1 | 3,647 | 17,122 | 3,513 |

**Приложение**

Листинг 1(тест Ферма)

**public static boolean** Fermat(Long n) {  
 Random r=**new** Random();  
 **int** a = r.nextInt(Math.*toIntExact*(n));  
 **while** (a>n || a==0&&a<n)  
 a=r.nextInt(Math.*toIntExact*(n));  
  
 **while** (n / a != 1)  
 a++;  
 BigInteger c = **new** BigInteger(String.*valueOf*(a));  
 **if** (c.modPow(BigInteger.*valueOf*(n - 1), BigInteger.*valueOf*(n)).equals(BigInteger.***ONE***)) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
}

Листинг 2(тест Соловэя-Штрассена)

**public static** Long Jacobi(Long a, Long b) {  
 **for** (**int** i = 2; i < b; i++) {  
 **if** (b % i == 0) {  
 System.***out***.println(**"Number "** + b + **" is composite"**);  
 **return** 0L;  
 }  
 }  
 Long r = 1L;  
 **if** (a < 0) {  
 a = -a;  
 **if** (b % 4 == 3)  
 r = -r;  
 }  
 **do** {  
  
 **int** t = 0;  
 **while** (a % 2 == 0) {  
 t++;  
 a /= 2;  
 }  
 **if** (t % 2 != 0) {  
 **if** (b % 8 == 3 || b % 8 == 5)  
 r = -r;  
 }  
 **if** (a % 4 == 3 && b % 4 == 3)  
 r = -r;  
 Long c = a;  
 a = b % c;  
 b = c;  
  
 }  
 **while** (a != 0);  
 **return** r;  
}  
  
**public static void** Solovei\_Shtrassen(Long n) {  
 System.***out***.println(**"Reliability parametr is 10"**);  
 **int** t = 10;  
 **int** end = 0;  
 **int** prime = 0;  
 System.***out***.print(**"Solovei-Shtrassen test:\n"** + n + **" is "**);  
 **while** (end != t) {  
 Long c = (**long**) (Math.*random*() \* (n - 2 - 2) + 2);  
 **for** (**int** i = 2; i < n; i++) {  
 **if** (n % i == 0) {  
 System.***out***.println(**"composite"**);  
 **return**;  
 }  
 }  
 **if** (n % c == 0 && n % 2 == 0) {  
 System.***out***.println(**"composite"**);  
 **return**;  
 } **else** {  
 Long r = *Jacobi*(c, n);  
  
 BigInteger s = **new** BigInteger(String.*valueOf*(c));  
 s = s.modPow(BigInteger.*valueOf*((n - 1) / 2), BigInteger.*valueOf*(n));  
 **if** (BigInteger.*valueOf*(r).equals(s))  
 prime++;  
 }  
 end++;  
  
 }  
  
 System.***out***.println(**"prime with probability "** + (1 - Math.*pow*(2, -prime)));  
}

Листинг 3(тест Миллера-Рабина)

**public static void** Miller\_Rabin(Long n) {  
 System.***out***.println(**"Number of rounds is 100"**);  
  
 **if** (n < 2 || n % 2 == 0) {  
 System.***out***.println(**"Wrong number"**);  
 **return** ;  
 } **else** {  
 System.***out***.print(**"Miller\_Rabin test:\n"**);  
 **for** (**int** i = 2; i < n; i++) {  
 **if** (n % i == 0) {  
 System.***out***.println(n + **" is a composite"**);  
 **return** ;  
 }  
 }  
 **int** k = 0;  
 Long t = n - 1;  
 **int** s = 0;  
 **while** (t % 2 == 0) {  
 t /= 2;  
 s++;  
 }  
 **int** prime = 0;  
 label:  
 **while** (k != 20) {  
 Long a = (**long**) (Math.*random*() \* (n - 2 - 2) + 2); BigInteger x = **new** BigInteger(String.*valueOf*(a));  
 x = x.modPow(BigInteger.*valueOf*(t), BigInteger.*valueOf*(n));  
 **if** (x.equals(BigInteger.***ONE***) || x.equals(BigInteger.*valueOf*(-1))) {  
 prime++;  
 k++;  
 **continue**;  
 } **else** {  
 **for** (**int** i = 0; i < s; i++) { x = x.modPow(BigInteger.*valueOf*(2), BigInteger.*valueOf*(n));  
 **if** (x.equals(BigInteger.***ONE***)) {  
 **break**;  
 }  
 **if** (x.equals(BigInteger.*valueOf*(-1))) {  
 prime++;  
 k++;  
 **continue** label;  
 }  
 }  
  
 }  
 k++;  
 }  
 System.***out***.print(n + **" is "**);  
 System.***out***.printf(**"prime with probability %.4f "**,1 - Math.*pow*(4, -prime));  
 System.***out***.println();  
 }

Листинг 3(Main class)

**public class** Main {  
 **public static void** main(String[] args) {  
 Scanner scanner = **new** Scanner(System.***in***);  
 System.***out***.println(**"Enter a number and choose needed test :\n"** +  
 **"1-Theorem of Ferma\n"** +  
 **"2-Solovei\_Shtrassen test\n"** +  
 **"3-Miller\_Rabin test\n"** );  
 System.***out***.println(**"Enter a number"**);  
  
  
 Long number = scanner.nextLong();  
 scanner.nextLine();  
 **while** (**true**) {  
  
 String command = scanner.nextLine();  
  
 **switch** (command) {  
 **case "1"**:  
 System.***out***.print(**"Ferma test:\n"** + number + **" is "**);  
 **if** (!*Fermat*(number))  
 System.***out***.println(**"composite"**);  
 **else** System.***out***.println(**"maybe prime"**);  
 **break**;  
 **case "2"**:  
 *Solovei\_Shtrassen*(number);  
 **break**;  
 **case "3"**:  
 *Miller\_Rabin*(number);  
 **break**;  
 **case "exit"**:  
 **return**;  
 **default**:  
 System.***out***.println(**"Wrong command,try again."**);  
 **break**;  
 }  
 }  
 }