Отчёт по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Кодже Лемонго Арман

Содержание

Список иллюстраций

# Цель работы

Изучение Вероятностные алгоритмы проверки чисел на простоту : алгоритмов Ферма, вычисления символа якоби, Соловэя-Штрассена, Миллера-Рабина.

# Теоретические сведения

Действительно, стойкость алгоритма RSA основана на возможности выбора простых чисел, состоящих из не менее 100 десятичных знаков. Поиск таких чисел последовательным перебором всех нечетных, начиная с некоторого стартового, и последующим «просеиванием» через решето Эратосфена весьма затруднителен. Проблему решают обходным маневром и решают «приближенно», а точнее — «вероятностно». Возможного кандидата на простое число подвергают испытанию серией однотипных и легко осуществимых тестов. Положительность результата хотя бы одного теста однозначно свидетельствует о том, что кандидат является числом составным; с другой стороны, отрицательный результат теста не дает абсолютной гарантии простоты кандидата, но свидетельствует о том, что вероятность его быть составным уменьшилась на определенную величину, скажем, в два раза. Тогда с увеличением количества отрицательных результатов все меньше шансов у испытуемого числа оказаться составным. Организовав серию испытаний из большого количества — например, 100 — тестов и получив все их результаты отрицательными, мы имеем право сказать, что кандидат является скорее всего («вроде бы») простым, с вероятностью не менее. Для таких чисел используют название вероятностно простые числа. Мы изложим здесь два способа организации упомянутой серии тестов, при этом тестируемое число будем обозначать.

## Алгоритм, реализующий тест Ферма

* Вход. Нечетное целое число .
* Выход. «Число n, вероятно, простое» или «Число n составное».

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

## Алгоритм вычисления символа якоби

1. Если НОД (a, b)≠1, выход из алгоритма с ответом 0.
2. Инициализация. r=1
3. Переход к положительным числам. Если a<0, то a=-a. Если b mod 4 = 3, то r=-r
4. Избавление от чётности. t=0. Цикл ПОКА a — чётное, t=t+1, a=a/2. Конец цикла. Если t — нечётное, то Если b mod 8 = 3 или 5, то r=-r.
5. Вычисление символа Якоби. 14 При перестановке аргументов больший заменяется на остаток от деления на меньший. Это возможно благодаря периодичности символа Якоби. Сложность алгоритма равна O(log a ⋅ log b) битовых операций.

## Алгоритм, реализующий Тест Соловэя-Штрассена

* Вход. Нечетное целое число .
* Выход. «Число n, вероятно, простое» или «Число n составное».

1. Выбрать случайное целое число .
2. Вычислить
3. При и результат: «Число n составное».
4. Вычислить символ Якоби
5. При результат: «Число n, вероятно, простое». В противном случае результат: «Число n составное».

## Алгоритм, реализующий Тест Миллера-Рабина.

* Вход. Нечетное целое число .
* Выход. «Число n, вероятно, простое» или «Число n составное».

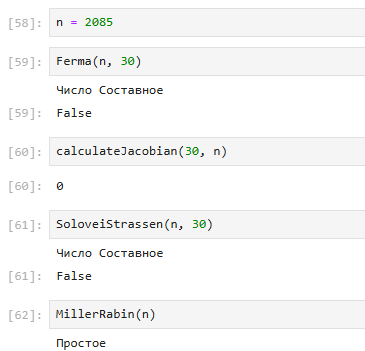
1. Представить в виде , где r - нечетное число
2. Выбрать случайное целое число .
3. Вычислить
4. При и выполнить действия
   * Положить
   * Если и то
     + Положить
     + При результат: «Число n составное».
     + Положить
   * При результат: «Число n составное».
5. Результат: «Число n, вероятно,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

# Выполнение работы

## Реализация алгоритмов на языке Python

import random  
  
- Алгоритм, реализующий тест Ферма  
  
def Ferma(n, test\_count):  
 for i in range(test\_count):  
 a = random.randint(2, n-1)  
 if ( a\*\*(n-1)%n != 1 ):  
 print("Число Составное")  
 return False  
 print("Вероятно, Простое")  
 return True  
  
- Алгоритм вычисления символа якоби   
  
def calculateJacobian(a, n):  
 if (a == 0):  
 return 0  
 ans = 1  
 if (a < 0):  
 a = -a  
 if (n%4 == 3):  
 ans = -ans  
 if (a == 1):  
 return ans  
 while (a):  
 if (a < 0):  
 a = -a  
 if (n%4 == 3):  
 ans = -ans  
 while (a%2 == 0):  
 a = a//2  
 if (n%8 == 3 or n%8 == 5):  
 ans = -ans  
 a, n = n, a  
 if (a%4 == 3 and n%4 == 3):  
 ans = -ans  
 a = a%n  
 if (a > n//2):  
 a = a - n  
 if(n == 1):  
 return ans  
 return 0  
  
- Алгоритм, реализующий Тест Соловэя-Штрассена  
  
def SoloveiStrassen(p, iterations):  
 if (p < 2):  
 print("Число Составное")  
 return False  
 if (p != 2 and p%2 == 0):  
 print("Число Составное")  
 return False  
 for i in range(iterations):  
 a = random.randrange(p - 1) + 1  
 jacobian = (p + calculateJacobian(a, p))%p  
 mod = modulo(a, (p - 1)/2, p)  
 if (jacobian == 0 or mod != jacobian):  
 print("Число Составное")  
 return False  
 return True  
  
- Алгоритм, реализующий Тест Миллера-Рабина   
  
def MillerRabin(n):  
 if n != int(n):  
 print("Число Составное")  
 return False  
 n = int(n)  
 if n == 0 or n == 1 or n == 4 or n == 6 or n == 8 or n == 9:  
 print("Число Составное")  
 return False  
 if n == 2 or n == 3 or n == 5 or n == 7:  
 print("Простое")  
 return True  
 s = 0  
 d = n - 1  
 while d%2 == 0:  
 d >>= 1  
 s += 1  
 assert(2\*\*s\*d == n-1)  
   
 def trial\_compose(a):  
 if pow(a, d, n) == 1:  
 print("Число Составное")  
 return False  
 for i in range(s):  
 if pow(a, 2\*\*i\*d, n) == n - 1:  
 print("Число Составное")  
 return False  
 print("Вероятно, Простое")  
 return True  
   
 for i in range(8):  
 a = random.randrange(2, n)  
 if trial\_compose(a):  
 print("Число Составное")  
 return False  
 print("Вероятно, Простое")  
 return True

## Контрольный пример



Работа алгоритмов

# Выводы

Изучили алгоритмы Ферма, вычисления символа якоби, Соловэя-Штрассена, Миллера-Рабина.

# Список литературы

1. [проверка чисел на простоту](hhttps://spravochnick.ru/informatika/algoritmizaciya/proverka_chisel_na_prostotu/)
2. [Исследование алгоритмов генерации простых чисел](https://moluch.ru/archive/90/18929/)
3. [Вероятностно простые числа](https://vmath.ru/vf5/crypto/pprime)