Отчет по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Бурдина Ксения Павловна

9 ноября 2023

Содержание

# 1 Цель работы

Целью данной работы является освоение вероятностных алгоритмов проверки чисел на простоту.

# 2 Задание

1. Изучить вероятностные алгоритмы проверки чисел на простоту.
2. Реализовать представленные алгоритмы.

# 3 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 числе 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 и проверяют условия алгоритма. Если число не проходит тест по основанию , то алгоритм выдает результат “Число составное”, и число действительно является составным [[1]](https://esystem.rudn.ru/pluginfile.php/2089869/mod_folder/content/0/%D0%A2%D1%80%D0%B0%D0%B4%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%88%D0%B8%D1%84%D1%80%D1%8B%20%D1%81%20%D1%81%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%BC%20%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC.pdf?forcedownload=1).

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 .

Схема вероятностного алгоритма проверки числа на простоту [[2]](https://esystem.rudn.ru/pluginfile.php/2089890/mod_folder/content/0/mathsec_lection11-asymmetric-cryptography.pdf?forcedownload=1):

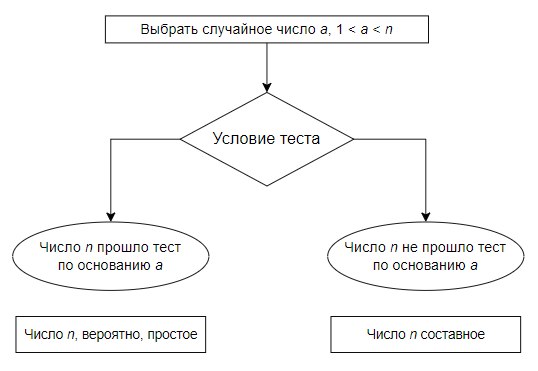
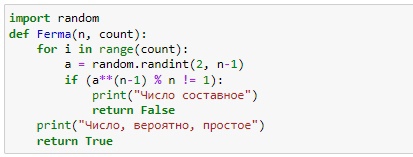


Схема проверки числа

# 4 Ход выполнения лабораторной работы

Для реализации вероятностых алгоритмов проверки чисел на простоту будем использовать среду JupyterLab. Выполним необходимую задачу.

1. Запишем алгоритм, реализующий тест Ферма, с помощью следующей функции:



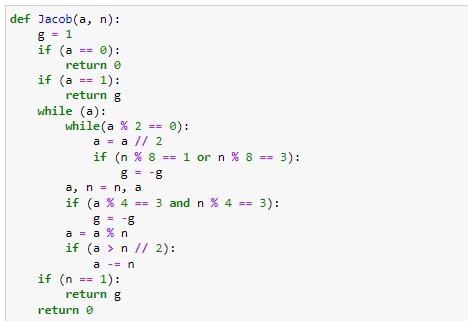
Тест Ферма

Здесь на вход подается нечетное целое число . Необходимо выполнить следующее:

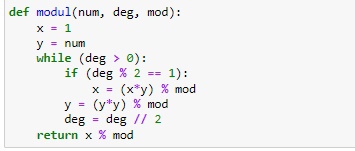
* выбрать случайное целое число ,
* вычислить ( )
* при результат: “Число , вероятно, простое”. В противном случае результат: “Число составное”.

По итогу при вызове функции мы получим результат проверки числа на простоту.

1. Реализуем алгоритм вычисления символа Якоби с помощью следующей функции:



Символ Якоби



Символ Якоби. Остаток

Здесь на вход поступает нечетное целое число , целое число , . Необходимо выполнить следующее:

* положить
* при результат:
* при результат:
* представить в виде , где число нечетное
* при четном положить , при нечетном положить , если ( ); положить , если ( )
* при результат:
* если ( ) и ( ), то
* положить ( ), и вернуться на шаг

По итогу при вызове функции мы получим символ Якоби .

1. Пропишем алгоритм, реализующий тест Соловэя-Штрассена, с помощью следующей функции:



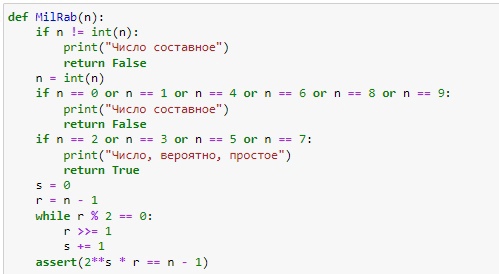
Тест Соловэя-Штрассена

Здесь на вход поступает нечетное целое число . Необходимо выполнить следующее:

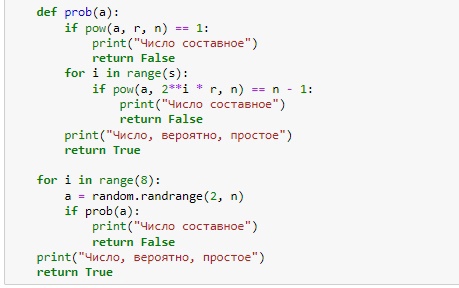
* выбрать случайное целое число ,
* вычислить ( )
* при и результат: “Число составное”
* вычислить *символ Якоби*
* при ( ) результат: “Число составное”. В противном случае результат: “Число , вероятно, простое”.

По итогу при вызове функции мы получим результат проверки числа на простоту.

1. Пропишем алгоритм, реализующий тест Миллера-Рабина, с помощью следующей функции:



Тест Миллера-Рабина



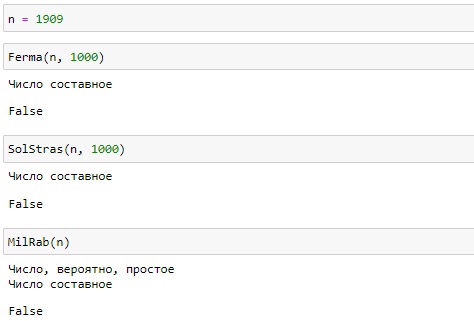
Тест Миллера-Рабина. Проба

Здесь на вход поступает нечетное целое число . Необходимо выполнить следующее:

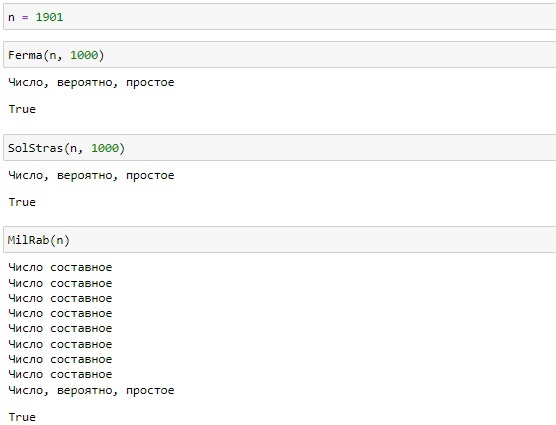
* представить в виде , где число нечетное
* выбрать случайное целое число ,
* вычислить ( )
* при и выполнить следующие действия:
  + положить
  + если и , то:
    - положить ( )
    - при результат: “Число составное”
    - положить
  + при результат: “Число составное”
* результат: “Число , вероятно, простое”

По итогу при вызове функции мы получим результат проверки числа на простоту.

1. Проверим работу алгоритмов:



Проверка работы алгоритмов 1



Проверка работы алгоритмов 2

Видим, что алгоритмы работают корректно, так как число является составным, а число - простым.

# 5 Листинг программы

import random  
def Ferma(n, count):  
 for i in range(count):  
 a = random.randint(2, n-1)  
 if (a\*\*(n-1) % n != 1):  
 print("Число составное")  
 return False  
 print("Число, вероятно, простое")  
 return True  
  
def Jacob(a, n):  
 g = 1  
 if (a == 0):  
 return 0  
 if (a == 1):  
 return g  
 while (a):  
 while(a % 2 == 0):  
 a = a // 2  
 if (n % 8 == 1 or n % 8 == 3):  
 g = -g  
 a, n = n, a  
 if (a % 4 == 3 and n % 4 == 3):  
 g = -g  
 a = a % n  
 if (a > n // 2):  
 a -= n  
 if (n == 1):  
 return g  
 return 0  
  
def modul(num, deg, mod):  
 x = 1  
 y = num  
 while (deg > 0):  
 if (deg % 2 == 1):  
 x = (x\*y) % mod  
 y = (y\*y) % mod  
 deg = deg // 2  
 return x % mod  
  
def SolStras(r, count):  
 if (r < 2):  
 print("Число составное")  
 return False  
 if (r != 2 and r % 2 == 0):  
 print("Число составное")  
 return False  
 for i in range(count):  
 a = random.randrange(r - 1) + 1  
 s = (r + Jacob(a, n)) % r  
 mod = modul(a, (r - 1) / 2, r)  
 if (s == 0 or mod != s):  
 print("Число составное")  
 return False  
 else:  
 print("Число, вероятно, простое")  
 return True  
  
def MilRab(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("Число, вероятно, простое")  
 return True  
 s = 0  
 r = n - 1  
 while r % 2 == 0:  
 r >>= 1  
 s += 1  
 assert(2\*\*s \* r == n - 1)  
  
 def prob(a):  
 if pow(a, r, n) == 1:  
 print("Число составное")  
 return False  
 for i in range(s):  
 if pow(a, 2\*\*i \* r, n) == n - 1:  
 print("Число составное")  
 return False  
 print("Число, вероятно, простое")  
 return True  
  
 for i in range(8):  
 a = random.randrange(2, n)  
 if prob(a):  
 print("Число составное")  
 return False  
 print("Число, вероятно, простое")  
 return True  
  
n = 1909  
Ferma(n, 1000)  
SolStras(n, 1000)  
MilRab(n)  
  
n = 1901  
Ferma(n, 1000)  
SolStras(n, 1000)  
MilRab(n)

# 6 Выводы

В ходе работы мы изучили и реализовали вероятностные алгоритмы проверки чисел на простоту.

# 7 Список литературы

1. Традиционные шифры с симметричным ключом [[1]](https://esystem.rudn.ru/pluginfile.php/2089869/mod_folder/content/0/%D0%A2%D1%80%D0%B0%D0%B4%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%88%D0%B8%D1%84%D1%80%D1%8B%20%D1%81%20%D1%81%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%BC%20%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC.pdf?forcedownload=1)
2. Методические материалы курса [[2]](https://esystem.rudn.ru/pluginfile.php/2089890/mod_folder/content/0/mathsec_lection11-asymmetric-cryptography.pdf?forcedownload=1)