Отчет по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Бурдина Ксения Павловна

9 ноября 2023

Содержание

Список иллюстраций

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 a - целое число. Числа $\pm 1, \pm a$ называются *тривиальными делителями* числа a.

Целое число $p \in Z/\{0\}$ называется *простым*, если оно не является делителем единицы и не имеет других делителей, кроме тривиальных. В противном случае число $p \in Z/\{-1, 0, 1\}$ называется *составным*.

Например, числа $\pm 2, \pm 3, \pm 5, \pm 7, \pm 11, \pm 13, \pm 17, \pm 19, \pm 23, \pm 29$ являются простыми.

Пусть $m \in N, m > 1$. Целые числа a и b называются сравнимыми по модулю m (обозначается $a \equiv b \; (mod \, m)$) если разность $a - b$ делится на m. Также эта процедура называется нахождением остатка от целочисленного деления a на b.

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 n вероятностным алгоритмом выбирают случайное число $a (1 < a < n)$ и проверяют условия алгоритма. Если число

n не проходит тест по основанию a, то алгоритм выдает результат “Число n составное”, и число n действительно является составным [1].

Если же n проходит тест по основанию a, ничего нельзя сказать о том, действительно ли число n является простым. Последовательно проведя ряд проверок таким тестом для разных a и получив для каждого из них ответ “Число n, вероятно, простое”, можно утверждать, что число n является простым с вероятностью, близкой к 1. После t независимых выполнений теста вероятность того, что составное число n будет t раз объявлено простым (вероятность ошибки), не превосходит $\frac{1}{2^t}$.

Схема вероятностного алгоритма проверки числа на простоту [2]:

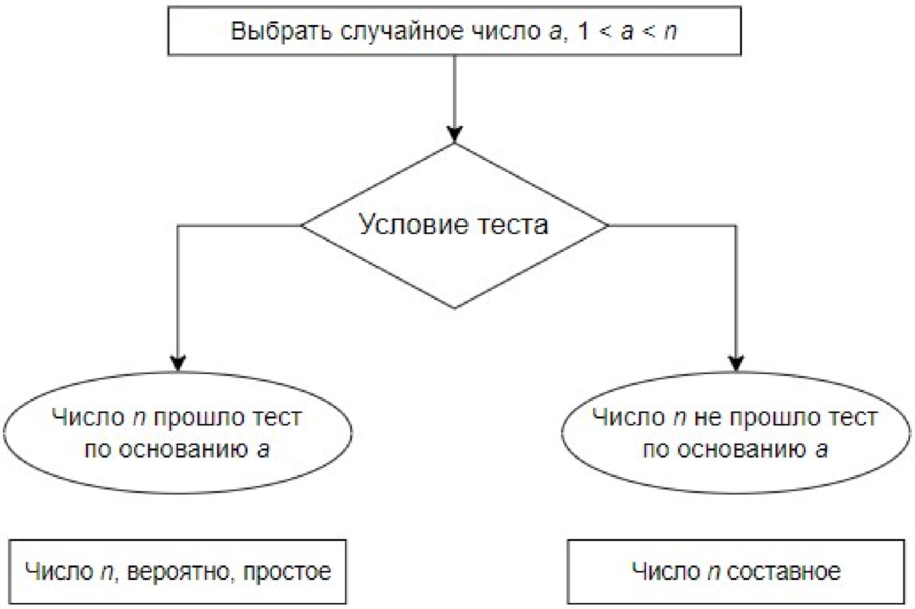

Схема проверки числа

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:

```
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
```

Тест Ферма

Здесь на вход подается нечетное целое число $n \geqslant 5$. Необходимо выполнить следующее:

- выбрать случайное целое число $a, 2 \leqslant a \leqslant n-2$
- вычислить $r \leftarrow a^{n-1}\ (mod\, n)$
- при $r = 1$ результат: "Число n, вероятно, простое". В противном случае результат: "Число n составное".

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

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

```
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
```

Символ Якоби

```
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
```

Символ Якоби. Остаток

Здесь на вход поступает нечетное целое число $n \geqslant 3$, целое число $a, 0 \leqslant a < n$. Необходимо выполнить следующее:

- положить $g \leftarrow 1$

- при $a = 0$ результат: 0
- при $a = 1$ результат: g
- представить a в виде $a = 2^k a_1$, где число a_1 нечетное
- при четном k положить $s \leftarrow 1$, при нечетном k положить $s \leftarrow 1$, если $n \equiv \pm 1 \ (mod\, 8)$; положить $s \leftarrow -1$, если $n \equiv \pm 3 \ (mod\, 8)$
- при $a_1 = 1$ результат: $g * s$
- если $n \equiv 3 \ (mod\, 4)$ и $a_1 \equiv 3 \ (mod\, 4)$, то $s \leftarrow -s$
- положить $a \leftarrow n \ (mod\, a_1)$, $n \leftarrow a_1$, $g \leftarrow g * s$ и вернуться на шаг 2

По итогу при вызове функции мы получим символ Якоби $\frac{a}{n}$.

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

```
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
```

Тест Соловэя-Штрассена

Здесь на вход поступает нечетное целое число $n \geqslant 5$. Необходимо выполнить следующее:

- выбрать случайное целое число a, $2 \leqslant a < n - 2$
- вычислить $r \leftarrow a^{\frac{n-1}{2}} \ (mod\, n)$

- при $r \neq 1$ и $r \neq n - 1$ результат: “Число n составное”
- вычислить *символ Якоби* $s \leftarrow \left(\frac{a}{n}\right)$
- при $r \equiv s \ (mod\ n)$ результат: “Число n составное”. В противном случае результат: “Число n, вероятно, простое”.

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

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

```
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)
```

Тест Миллера-Рабина

```python
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
```

Тест Миллера-Рабина. Проба

Здесь на вход поступает нечетное целое число $n \geqslant 5$. Необходимо выполнить следующее:

- представить $n-1$ в виде $n-1 = 2^s r$, где число r нечетное
- выбрать случайное целое число a, $2 \leqslant a < n-2$
- вычислить $y \leftarrow a^r \ (mod\, n)$
- при $y \neq 1$ и $y \neq n-1$ выполнить следующие действия:
  - положить $j \leftarrow 1$
  - если $j \leqslant s-1$ и $y \neq n-1$, то:
    - положить $y \leftarrow y^2 \ (mod\, n)$
    - при $y = 1$ результат: "Число n составное"
    - положить $j \leftarrow j+1$
  - при $y \neq n-1$ результат: "Число n составное"
- результат: "Число n, вероятно, простое"

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

5. Проверим работу алгоритмов:

```
n = 1909
```

```
Ferma(n, 1000)
```

```
Число составное

False
```

```
SolStras(n, 1000)
```

```
Число составное

False
```

```
MilRab(n)
```

```
Число, вероятно, простое
Число составное

False
```

Проверка работы алгоритмов 1

```
n = 1901
```

```
Ferma(n, 1000)
```

```
Число, вероятно, простое

True
```

```
SolStras(n, 1000)
```

```
Число, вероятно, простое

True
```

```
MilRab(n)
```

```
Число составное
Число составное
Число составное
Число составное
Число составное
Число составное
Число составное
Число составное
Число, вероятно, простое

True
```

Проверка работы алгоритмов 2

Видим, что алгоритмы работают корректно, так как число 1909 является составным, а число 1901 - 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]

2. Методические материалы курса [2]