Отчет по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Бармина Ольга Константиновна

2024 September 7th

Содержание

List of Figures

1 Цель работы

Цель данной работы: научиться реализовывать алгоритмы проверки чисел на простоту.

2 Задание

1. Реализовать алгоритмы проверки чисел на простоту.

3 Теоретическое введение

Пусть a - целое число. Числа ‡1, ‡a называются тривиальными делителями числа a. Целое число p €Z/{0) называется простым, если оно не является делителем единицы и не имеет других делителей, кроме тривиальных. В противном случае число p €Z/{-1, 0, 1) называется составным.

4 Выполнение лабораторной работы

1. Прописываем функцию для алгоритма теста Ферма (рис. fig. 4.1).

```
a = 29
b = 49
```

```
def ferm(a, n):
    r = (a**(n-1))%n
    if r == 1:
        return 'probably, prime'
    else:
        return 'not prime'
```

```
ferm(8,a)
```

```
'probably, prime'
```

```
ferm(8,b)
```

```
'not prime'
```

Figure 4.1: Программная реализация алгоритма теста Ферма.

2. Прописывается функция для алгоритма вычисления символа Якоби. (рис.

fig. 4.2).

```
from sympy import primefactors

def jacobi(a, n):
    g = 1
    while True:
        if a == 0:
            return 0
        elif a == 1:
            return g
        else:
            k = primefactors(a)[0]
            if len(primefactors(a)) == 1:
                a1 = a
            else:
                a1 = primefactors(a)[1]
            if k%2 == 1:
                if (n-1)%8 == 0 or (n+1)%8 == 0:
                    s = 1
                if (n-3)%8 == 0 or (n+3)%8 == 0:
                    s = -1
            else:
                s = 1
            if a1 == 1:
                return g*s
            if (n-3)%4 == 0 or (a1-3)%4 == 0:
                s = -s
            a = n%a1
            n = a1
            g = g*s
```

```
jacobi(8, a)
```

```
-1
```

```
jacobi(8, b)
```

```
1
```

Figure 4.2: Алгоритм вычисления символа Якоби

3. Программная реализация алгоритма Соловэй-Штрассена. (рис. fig. 4.3).

```
def sol_st(a, n):
    r = (a**((n-1)/2))%n
    if r != 1 and r != n-1:
        return 'not prime'
    s = jacobi(a, n)
    if (r-s)%n != 0:
        return 'not prime'
    else:
        return 'probably prime'
```

```
sol_st(8, a)
```

```
'probably prime'
```

```
sol_st(8, b)
```

```
'not prime'
```

Figure 4.3: Программная реализация алгоритма Соловэй-Штрассена

4. Программная реализация алгоритма Миллера-Рабина. (рис. fig. 4.4).

```python
def mil_rab(a, n):
    s = primefactors(n-1)[0]
    if len(primefactors(n-1)) == 1:
        r = n-1
    else:
        r = primefactors(n-1)[1]
    y = (a**r)%n
    if y != 1 and y != n-1:
        j = 1
        while j <= s-1 and y != n-1:
            y = (y**2)%n
            if y == 1:
                return 'not prime'
            j += 1
        if y != n-1:
            return 'not prime'
    return 'probably prime'
```

```python
mil_rab(8, a)
```

```
'probably prime'
```

```python
mil_rab(8, b)
```

```
'not prime'
```

Figure 4.4: Программная реализация алгоритма Миллера-Рабина.

5 Выводы

В ходе работы были реализованы алгоритмы проверки чисел на простоту.