

Отчёт по лабораторной работе 5

МОЗИИБ

Папикян Гагик Тигранович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Алгоритм Соловья	7
3.2	Алгоритм Рабина	7
3.3	Алгоритм Ферма	8
4	Выполнение лабораторной работы	9
5	Выводы	14

List of Figures

3.1	Алгоритм	7
4.1	Выполнение лабораторной работы	13

List of Tables

1 Цель работы

Познакомиться с алгоритмами поиска Наибольшего Общего Делителя(НОД)

2 Задание

- 1) Реализовать алгоритм теста Ферма
- 2) Реализовать алгоритм Соловья
- 3) Реализовать алгоритм Рабина

3 Теоретическое введение

3.1 Алгоритм Соловья

Тест Соловья — Штрассена — вероятностный тест простоты, открытый в 1970-х годах Робертом Мартином Соловеем совместно с Фолькером Штрассеном. Тест всегда корректно определяет, что простое число является простым, но для составных чисел с некоторой вероятностью он может дать неверный ответ. Основное преимущество теста заключается в том, что он, в отличие от теста Ферма, распознает числа Кармайкла как составные.

```
Вход:  $n > 2$ , тестируемое нечётное натуральное число;  
       $k$ , параметр, определяющий точность теста.  
Выход: составное, означает, что  $n$  точно составное;  
       вероятно простое, означает, что  $n$  вероятно является простым.  
  
for  $i = 1, 2, \dots, k$ :  
     $a$  = случайное целое от 2 до  $n - 1$ , включительно;  
    если  $\text{НОД}(a, n) > 1$ , тогда:  
        вывести, что  $n$  — составное, и остановиться.  
    если  $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$ , тогда:  
        вывести, что  $n$  — составное, и остановиться.  
  
иначе вывести, что  $n$  — простое с вероятностью  $1 - 2^{-k}$ , и остановиться.
```

Figure 3.1: Алгоритм

3.2 Алгоритм Рабина

Тест Миллера — Рабина — вероятностный полиномиальный тест простоты. Тест Миллера — Рабина, наряду с тестом Ферма и тестом Соловья — Штрассена,

позволяет эффективно определить, является ли данное число составным. Однако, с его помощью нельзя строго доказать простоту числа. Тем не менее тест Миллера — Рабина часто используется в криптографии для получения больших случайных простых чисел.

3.3 Алгоритм Ферма

Тест простоты Ферма в теории чисел — это тест простоты натурального числа n , основанный на малой теореме Ферма.

При использовании алгоритмов быстрого возведения в степень по модулю время работы теста Ферма для одного a оценивается как $O(\log^2 n \times \log \log n \times \log \log \log n)$, где n — проверяемое число. Обычно проводится несколько проверок с различными a .

4 Выполнение лабораторной работы

Был написан следующий скрипт на javascript

```
function fermaTest(n){
    // if(n<2) return false
    // if(n in [2,3]) return true
    for(let i =0;i<200;i++){
        const a = Math.random() * (n-4) + 2
        const r = Math.pow(a, n-1) % n
        if(r === 1) return true
    }
    return false
}

// let result = ''
// for(let i = 5;i<25;i++){
//     result += `${i}(${fermaTest(i)}), `
// }
// console.log(result)

function jacobi(n, k){
    // assert(k > 0 && k % 2 === 1)
    n = n % k
    let t = 1
```

```

while (n !== 0) {
  while (!n%2){
    n = n / 2
    let r = k % 8
    if(r === 3 || r === 5) t = -t
  }
  [n, k] = [k, n]
  if (n % 4 === 3 && k % 4 === 3) t = -t
  n = n % k
}

if ( k === 1)
  return t
else
  return 0
}

function solovoyStrassen(p, iteration = 200) {
  for(let i = 0; i<iteration; i++){
    let r = Math.floor(Math.random()*2 );

    const a = r % (p - 1) + 1
    let j = (p + jacobi(a, p)) % p
    let mod = Math.pow(a, Math.floor((p-1)/2)) % p

    if(j === 0 || mod !== j) return false
  }
  return true
}

```

```
}
```

```
// let result = ''  
// for(let i = 5;i<25;i++){  
//     result += `${i}(${solovoyStrassen(i)}), `  
// }  
// console.log(result)
```

```
function millerRibben(n, k=100) {  
    if (n % 2 === 0) return false  
  
    var s = 0, d = n - 1;  
    while (d % 2 === 0) {  
        d /= 2;  
        ++s;  
    }  
  
    WitnessLoop: do {  
        //A base between 2 and n - 2  
        // let x = Math.random() * (n-4) + 2  
        var x = Math.pow(2 + Math.floor(Math.random() * (n - 3)), d) % n;  
  
        if (x === 1 || x === n - 1)  
            continue;  
  
        for (var i = s - 1; i--;) {  
            x = x * x % n;
```

```

        if (x === 1)
            return false ;
        if (x === n - 1)
            continue WitnessLoop;
    }

    return false ;
} while (--k);

return true ;
}

let result = ''
for(let i = 5; i < 25; i++){
    result += `${i}(${millerRibben(i)}), `
}
console.log(result)

```

Результат исполнения скрипта приведен на рисунке 1 (рис. 4.1)

The image shows a code editor with a JavaScript file named `index.js` and a terminal window. The code implements a Miller-Rabin primality test. It starts by defining a function `isProbablyPrime(n)` that uses a `WitnessLoop` to test a random base `a` between 2 and $n-2$. The test involves computing $x = a^{(n-1)/2} \pmod n$ and checking if $x \equiv 1$ or $x \equiv -1$. If not, it checks for non-trivial square roots of 1. The function returns `true` if the number passes the test for a given base, and `false` otherwise. The main part of the code is a loop that tests the first 25 numbers (from 5 to 29) and prints the results. The terminal window shows the output of the program, which lists the primality of each number from 5 to 29.

```
lab5 > JS index.js > @jacob  
64 var s = 0, d = n - 1;  
65 while (d % 2 === 0) {  
66   d /= 2;  
67   ++s;  
68 }  
69  
70 WitnessLoop: do {  
71   //A base between 2 and n - 2  
72   // let x = Math.random() * (n-4) + 2  
73   var x = Math.pow(2 + Math.floor(Math.random() * (n - 3)), d) % n;  
74  
75   if (x === 1 || x === n - 1)  
76     continue;  
77  
78   for (var i = s - 1; i--;) {  
79     x = x * x % n;  
80     if (x === 1)  
81       return false;  
82     if (x === n - 1)  
83       continue WitnessLoop;  
84   }  
85  
86   return false;  
87 } while (--k);  
88  
89 return true;  
90 }  
91  
92 let result = ''  
93 for(let i = 5; i <= 29; i++){  
94   result += `${i}(${MillerRabin(i)}), `;  
95 }  
96 console.log(result)  
97  
98  
99
```

Terminal output:

```
lab5 -- zsh -- 86x20  
robably prime), 24(24 is probably NOT prime),  
gaglopapinni@gaglopapinni-osx lab5 % node index.js  
5(true), 6(6 is probably NOT prime), 7(true), 8(8 is probably NOT prime), 9(false), 10  
(10 is probably NOT prime), 11(true), 12(12 is probably NOT prime), 13(true), 14(14 is  
probably NOT prime), 15(false), 16(16 is probably NOT prime), 17(true), 18(18 is prob  
ably NOT prime), 19(true), 20(20 is probably NOT prime), 21(false), 22(22 is probably  
NOT prime), 23(true), 24(24 is probably NOT prime),  
gaglopapinni@gaglopapinni-osx lab5 % node index.js  
5(true), 6(false), 7(true), 8(false), 9(false), 10(false), 11(true), 12(false), 13(tru  
e), 14(false), 15(false), 16(false), 17(true), 18(false), 19(true), 20(false), 21(fals  
e), 22(false), 23(true), 24(false),  
gaglopapinni@gaglopapinni-osx lab5 % node index.js  
5(true), 6(false), 7(true), 8(false), 9(false), 10(false), 11(true), 12(false), 13(tru  
e), 14(false), 15(false), 16(false), 17(true), 18(false), 19(true), 20(false), 21(fals  
e), 22(false), 23(true), 24(false),  
gaglopapinni@gaglopapinni-osx lab5 % node index.js  
5(true), 6(false), 7(true), 8(false), 9(false), 10(false), 11(true), 12(false), 13(tru  
e), 14(false), 15(false), 16(false), 17(true), 18(false), 19(true), 20(false), 21(fals  
e), 22(false), 23(true), 24(false),  
gaglopapinni@gaglopapinni-osx lab5 %
```

Figure 4.1: Выполнение лабораторной работы

5 Выводы

Были реализованы приведенные алгоритмы, и продемонстрирован результат их выполнения