Отчёт по лабораторной работе 5

МОЗИиИБ

Папикян Гагик Тигранович

Содержание

List of Figures

List of Tables

1 Цель работы

Познакомиться с алгоритмами поиска Наибольшего Общего Делителя(НОД)

2 Задание

1) Реализовать алгоритм теста Ферма
2) Реализовать алгоритм Соловэя
3) Реализовать алгоритм Рабина

3 Теоретическое введение

3.1 Алгоритм Соловэя

Тест Соловея — 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 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 Соловея — 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(log2n × log log n × 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)

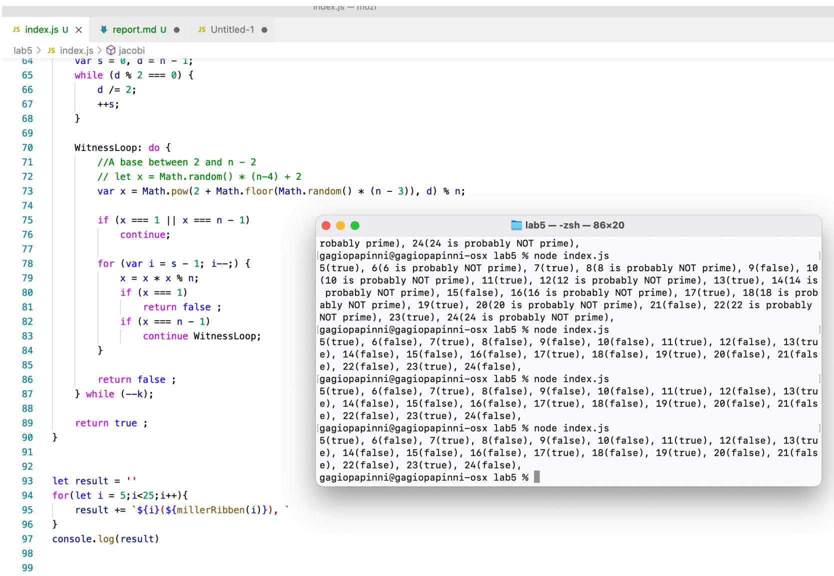

Figure 4.1: Выполнение лабораторной работы

5 Выводы

Были реализованы приведенные алгоритмы, и продемонстирован результат их выполнения