КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

Звіт

Із дисципліни «Хмарні обчислення»

Лабораторна робота на тему

ПЕРЕВІРКА ПРОСТОТИ ЧИСЛА ЙМОВІРНИСНИМ ТЕСТОМ СОЛОВЕЯ-ШТРАССЕНА

Виконав студент 4-го курсу

Шевченко Максим Олексійович

КИЇВ 2021

Проєкт доступний за посиланням:

https://github.com/Gurdel/my-parcs-java

Постановка задачі

Задано число N та кількість ітерацій k. Перевірити число на простоту алгоритмом Соловея-Штрассена.

Теоретичні відомості

Просте число — це натуральне число, яке має рівно два різних натуральних дільники (лише 1 і саме число). Решту чисел, окрім одиниці, називають складеними. Таким чином, всі натуральні числа, більші від одиниці, розбивають на прості і складені. Теорія чисел вивчає властивості простих чисел. В теорії кілець простим числам відповідають незвідні елементи.

Такі тести простоти, як решето Ератосфена, решето Сундарама та решето Аткіна, дають прості способи складання початкового списку простих чисел до певного значення. Однак на практиці замість отримання списку простих чисел найчастіше потрібно перевірити, чи є дане число простим. Алгоритми, які вирішують це завдання, називають тестами простоти. Існує безліч поліноміальних тестів простоти, але більшість з них є стохастичними (наприклад, тест Міллера — Рабина) і використовуються для потреб криптографії. Тільки в 2002 році було доведено, що завдання перевірки на простоту в загальному вигляді можна розв'язати за поліноміальний час, але запропонований детермінований алгоритм має досить велику складність, що ускладнює його застосування на практиці. Для деяких класів чисел існують спеціалізовані ефективні тести простоти. Наприклад, для перевірки на простоту чисел Мерсенна використовують тест Люка — Лемера, а для перевірки на простоту чисел Ферма — тест Пепіно.

Тест Соловея — Штрассена — імовірнісний тест простоти, відкритий у 1970-х роках Робертом Мартіном Соловеем спільно з Фолькером Штрассеном. Тест завжди коректно визначає, що просте число є простим, але для складених чисел з деякою ймовірністю він може дати неправильну відповідь. Основна перевага тесту полягає в тому, що він, на відміну від тесту Ферма, розпізнає числа Кармайкла як складені.

Символ Якобі — в теорії чисел узагальнення символу Лежандра для довільних додатних непарних цілих чисел. Якщо розклад n на прості множники має вигляд $p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k}$, то символ Якобі рівний:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{c_1} \left(\frac{a}{p_2}\right)^{c_2} \cdots \left(\frac{a}{p_k}\right)^{c_k}$$, де в правій частині є звичайні символи Лежандра. Якщо n є простим числом то символ Якобі дорівнює символу Лежандра.

Символом Лежандра називається мультиплікативна функція, що використовується в теорії чисел. Названа на честь французького математика Адрієна-Марі Лежандра.

Нехай a деяке ціле число і p просте число. Символ Лежандра $\left(\frac{a}{p}\right)$ визначається таким чином:

0, якщо a ділиться на p; 1, якщо a є квадратичним лишком за модулем p, тобто існує таке ціле x, що $x^2 = a \pmod{p}$; -1, якщо a є квадратичним нелишком за модулем p.

Алгоритм, його реалізація

Алгоритм на псевдокоді може бути записаний наступним чином:

Ввід: $n > 2$, непарне натуральне число, що тестується;
k, параметр, що визначає точність тесту.
Вивід: *складене*, означає, що n точно складене;
ймовірно просте, означає, що n ймовірно є простим.

for $i = 1, 2, \ldots, k$:
a = випадкове ціле від 2 до $n-1$, включно;
якщо НСД(a, n) > 1, **тоді:**
вивести, що n – складене, і **зупинитися.**
якщо $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$, **тоді:**
вивести, що n – *складене*, і **зупинитися.**

інакше вивести, що n – *просте* *зі ймовірністю* $1 - 2^{-k}$, і **зупинитися.**

Точність алгоритму 2^{-k}, де k – кількість ітерацій.

Реалізацію алгоритму наведено на скріншотах:

```
    public boolean isPrime(BigInteger n, int iteration)
{
    /** base case **/
    if (n.equals(ZERO) || n.equals(ONE))
        return false;
    /** base case - 2 is prime **/
    if (n.equals(TWO))
        return true;
    /** an even number other than 2 is composite **/
    if (n.mod(TWO).equals(ZERO))
        return false;

    Random rand = new Random();
    for (int i = 0; i < iteration; i++)
    {
        BigInteger r = getRandomBigInteger();
        BigInteger a = r.mod(n.subtract(ONE)).add(ONE);
        BigInteger jacobian = (n.add(Jacobi(a, n))).mod(n);
        BigInteger mod = a.modPow(n.subtract(ONE).divide(TWO), n);
        if(jacobian.equals(ZERO) || !mod.equals(jacobian))
            return false;
    }
    return true;
}
```

```java
    /** Function to calculate jacobi (a/b) **/
public BigInteger Jacobi(BigInteger a, BigInteger b)
{
    if (b.compareTo(ZERO) <= 0 || b.mod(TWO).equals(ZERO))
        return ZERO;
    BigInteger j = ONE;
    if (a.compareTo(ZERO) < 0)
    {
        a = ZERO.subtract(a);
        if (b.mod(FOUR).equals(THREE))
            j = ZERO.subtract(j);
    }
    while (!a.equals(ZERO))
    {
        while (a.mod(TWO).equals(ZERO))
        {
            a = a.divide(TWO);
            if (b.mod(EIGHT).equals(THREE) || b.mod(EIGHT).equals(FIVE))
                j = ZERO.subtract(j);
        }

        BigInteger temp = a;
        a = b;
        b = temp;

        if (a.mod(FOUR).equals(THREE) && b.mod(FOUR).equals(THREE))
            j = ZERO.subtract(j);
        a = a.mod(b);
    }
    if (b.equals(ONE))
        return j;
    return ZERO;
}
/** Function to check if prime or not **/
public static BigInteger getRandomBigInteger() {
    Random rand = new Random();
    BigInteger result = new BigInteger(100, rand);
    return result;
}
```

Конкретну реалізацію всього проєкту можна знайти за посиланням:
https://github.com/Gurdel/my-parcs-java

Тестування

Тестування проводилося на персональному комп'ютері та за допомогою веб-сервісів на одному й двох воркерах. Так як не прості числа визначаються на одній із ітерацій, то використання їх для тестів не дасть стабільного прогнозованого результату по часу виконання алгоритму. Тому було обрано два простих числа: 4776913109852041418248056622882488319 і 5311379928167670986895882065524686273295931177270319231994441382004035598608522427391625022652292856688893294862465010153465793376527072394095199787665873519438312708353932190317281 27.

Результати тестування наведені в таблиці:

number	iterations	pc	time
4,78E+36	1000	my pc	0.2707029
		cloud 1	0.313757558
		cloud 2	0.373903158
	99999	my	5.2066753
		cloud 1	7.537769223
		cloud 2	8.826535413
5,3E+182	1000	my	0.7946785
		cloud 1	0.57704676
		cloud 2	0.709074977
	99999	my	29.3232008
		cloud 1	27.649709514
		cloud 2	14.034734985
	999999	my	337.2472933
		cloud 1	268.169907031
		cloud 2	133.170618967

Висновок: застосування технології ПАРКС, паралельних обчислень та хмарних технологій дає змогу значно пришвидшити виконання алгоритму на великих вхідних даних. При малих вхідних даних виконання алгоритму на комп'ютері дало кращі результати по швидкості виконання. Це пояснюється необхідністю

додаткових затрат на розпаралелення алгоритму, які переважають можливу вигоду від паралельних обчислень малого набору даних. Також можна помітити, виконання алгоритму одним воркером на деяких вхідних даних є швидшим за відповідний час виконання на комп’ютері. Це є наслідком використання потужних процесорів на серверах Google.