COMPARACIÓN DE ALGORITMOS DE TEST DE PRIMALIDAD

Los test de primalidad, es decir algoritmos que nos permiten comprobar si un número es primo o compuesto, son usados en múltiples ámbitos principalmente en la seguridad. Cabe señalar que existen algoritmos determinísticos y probabilísticos, los primeros apuntan a demostrar fehacientemente si un número es primo, y los segundos tan solo con probabilidades pero son mucho más rápidos en el proceso.

En este estudio se compararon 3 algoritmos, Solovay-Strassen, Miller-Rabin y Fermat, los cuales se aplican probabilísticamente.

Test de Solovay-Strassen

Un algoritmo probabilístico que certifica la composición más que la primalidad. Este test ya no se usa en general ya que el test de Miller-Rabin lo superó.

La probabilidad de error es de $\frac{1}{2}$.
Complejidad es $O(\log n)^3$.

Usa el Símbolo de Jacobi que es una mejora del símbolo de Legendre:

Símbolo de jacobi:

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$$

$$\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$$

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$$

$$\left(\frac{a}{n}\right) = \begin{cases} -\left(\frac{n}{a}\right) & \text{si } a \equiv n \equiv 3 \pmod 4 \\ \left(\frac{n}{a}\right) & \text{en otro caso} \end{cases}$$

La parte fundamental del algoritmo:

Se basa principalmente en:

$$\left(\frac{a}{n}\right) \neq a^{\frac{n-1}{2}} \pmod p$$

Se busca que el símbolo de Jacobi y la exponenciación modular sean diferentes.

Pseudo-algoritmo:

Entrada: **“n”** entero.
Salida: **“n”** es primo o **“n”** es compuesto.

Solovay-Strassen(n)

1) Se elige al azar un número entre 2 y n - 1.
2) Se calcula si el mcd(a, n) es diferente de 1
    a) Si lo es, está compuesto.
3) Si jacobi(a, n) es diferente de $a^{(n-1)/2}$ mod n
    a) Si lo es, está compuesto.

4) Si no se aplica en lo anterior, probablemente es primo.

Seguimiento:

Cuando n = 17:

Se escoge 2 como “número aleatorio”:
mcd(2, 17):
17 = 8 * 2 + 1
= 1

jacobi(2, 17) = -1 (mod 17) = 16

2^{8} (mod 17) = 16

Son iguales entonces, entonces 17 probablemente es primo

Implementación en C++:

```cpp
bool solovay_strassen(ZZ n)
{
    if ((n & 1) == 0)
        return false;

    ZZ a(2);
    for (int i = 0; i < 10; i++)
    {
        if(euclides_clasico(a, n) != 1){
            return false;
        }
        ZZ r = modulo(jacobi(a, n), n);

        ZZ _r = exponenciacion_mod(a, (n-1)/2,n);

        if (r != _r){
            return false;
        }
        do{
            a = RandomBnd(n);
        }
        while(a < 2);
    }
    return true;
}
```

Test de Miller-Rabin

El test probabilístico de Miller-Rabin es una extensión del test de Fermat, es uno de los más usados.

Se basa en la comprobación de diferentes bases para probarlo.

Probabilidad de error es de ¼.

La complejidad O((log n)³)

La parte fundamental del algoritmo:

Se descompone **n** en $\mathbf{n^s \cdot \mathit{d} + 1}$, donde d es impar, son enteros positivos.

Diremos que es probablemente primo si se cumple alguna de:

$$a^d \equiv 1 \pmod{n}$$

$$a^{2^r \cdot d} \equiv -1 \pmod{n}$$ para algún $0 \leq r < s$.

- Las únicas raíces cuadradas del 1 modulo n son 1 y -1 (mod n)

Pseudo-algoritmo:

Entrada: **“n”** entero.
Salida: **“n”** es primo o **“n”** es compuesto.

Miller-Rabin(n)

1) Descomponer **n - 1** en **2**s * t**, donde **t** es impar.
2) Repetir **k** veces:
    a) Un entero **a,** aleatorio entre **a** y **n - 1**.
    b) Un entero **x = a**t (mod n)**
    c) Si **x** es igual a **1**, o, **x** es igual **n - 1**
        i) Continuar
    d) Repetir **s - 1** veces:
        i) **x = x**2 (mod n)**

ii) Si **x** es igual a **1**:
   (1) Es un número compuesto.
iii) Si **x** es igual a **n - 1**:
   (1) Romper
e) Si **x** es diferente de **n - 1:**
   i) Es un número compuesto:
3) Probablemente es un número primo.

Seguimiento:

Cuando **n** es igual a 19:

Descomponiendo **n - 1** en **2**s * t.**
s = 1 y t = 9
18 = 21 * 9**

Repitiendo una sola vez:
eligiendo a **2** como base **a**

Repitiendo s - 1 veces (0).

18 es igual a 18

Por lo tanto es 19 probablemente es primo.

Implementación en C++:

```cpp
bool miller_rabin(ZZ n){

    if ((n & 1) == 0)
        return false;
    ZZ s(0);
    ZZ t = n - 1;
    while ((t & 1) == 0) {
        s++;
        t >>= 1;
    }
    ZZ a(2);
    for (int i = 0; i < 10; i++){

        ZZ x = exponenciacion_mod(a, t, n);
        if (x == 1 || x == (n-1))
            continue;
        for (ZZ r(0); r < (s- 1); r++){
            x = exponenciacion_mod(x, to_ZZ(2), n);
            if(x == 1){
                return false;
            }
            else if(x == n - 1)
                break;
        }
        if(x != n - 1 )
            return false;
        ZZ a = RandomBnd(n-3) + 3;
    }

    return true;
}
```

Test de Fermat

Este algoritmo probabilístico es simple, y es la base de otros teoremas, como algoritmos, en este caso se usa de forma básica, y no es tan empleado.

Parte fundamental:

Si **n** es primo y **a** es coprimo con **n**:

$$a^{n-1} \equiv 1 \pmod{n}$$

Pseudo-algoritmo:

Entrada: **“n”** entero.
Salida: **“n”** es primo o **“n”** es compuesto.

Fermat(n):

1) tomando como base **a = 2**
2) Si **a ** n - 1 (mod n)** es diferente de **1**
    a) Es un primo compuesto
3) Es probablemente un primo.

Seguimiento:

Cuando **n** es igual a 19:

eligiendo a **2** como base **a**

2 ** 18 (mod 19) = 1

Como 1 no es diferente de 1 :

Por lo tanto es 19 probablemente es primo.

Implementación en C++:

```cpp
bool fermat(ZZ n){

    if ((n & 1) == 0)
        return false;
    ZZ a(2);
    for (int i = 0; i < 20; i++){

        if (exponenciacion_mod(a, n - 1, n) != 1){
            return false;
        }
        a = RandomBnd(n-2) + 2;

    }
    return true;
}
```

Comparación de algoritmos :

```
solovay 8bits
Tiempo: 0.000065 sec

miller 8bits
Tiempo: 0.000011976 sec

fermat 8bits
Tiempo: 0.000001605 sec

--------------------------------
solovay 16bits
Tiempo: 0.000005322 sec

miller 16bits
Tiempo: 0.000016749 sec

fermat 16bits
Tiempo: 0.000002607 sec

--------------------------------
solovay 32bits
Tiempo: 0.000000163 sec

miller 32bits
Tiempo: 0.000000164 sec

fermat 32bits
Tiempo: 0.000000151 sec

--------------------------------
```

```
-----------------------------------
solovay 64bits
Tiempo: 0.000000106 sec

miller 64bits
Tiempo: 0.000000104 sec

fermat 64bits
Tiempo: 0.000000098 sec

-----------------------------------
solovay 128bits
Tiempo: 0.000026011 sec

miller 128bits
Tiempo: 0.000022918 sec

fermat 128bits
Tiempo: 0.000022714 sec

-----------------------------------
solovay 512bits
Tiempo: 0.000000182 sec

miller 512bits
Tiempo: 0.000000190 sec

fermat 512bits
Tiempo: 0.000000136 sec

-----------------------------------
```

```
------------------------
solovay 1024bits
Tiempo: 0.001102376 sec

miller 1024bits
Tiempo: 0.001139183 sec

fermat 1024bits
Tiempo: 0.001055212 sec

------------------------
solovay 2048bits
Tiempo: 0.006712559 sec

miller 2048bits
Tiempo: 0.006653939 sec

fermat 2048bits
Tiempo: 0.006543491 sec
```

Conclusiones

En cuanto a tiempo se refiere, se podría considerar a Fermat como un claro ganador. Pero también hay que considerar que es más probable que arroje resultados erróneos, por lo que Miller-Rabin tendrá mejor efecto en este caso.

Entonces podríamos decir que Miller-Rabin es el ganador.

Enlace de Github:

https://github.com/Brigham-CG/Brigham_CaceresGutierrez/tree/main/PrimalityTest

Bibliografía:

Breve Reseña sobre la Hipótesis de Riemann, Primalidad y el algoritmo AKS (José de Jesús Angel Angel Guillermo Morales-Luna, 2005).

Section 31.8: Primality testing». *Introduction to Algorithms* (Second edición). MIT