Laboratorio 1: Unidad 1

Christian Camilo Rivadeneira
A00354996
Juan Andrés Orozco Nuñez
A00355202

Universidad ICESI - Facultad de Ingeniería Sistemas
Santiago de Cali - Febrero 2 del 2019
2020

Desarrollo Laboratorio AED No.1

1. IDENTIFICACIÓN DEL PROBLEMA.

La empresa para la cual trabajamos ha venido desarrollando un plan en el cual pueda ofrecer una mejor seguridad a cada uno de los sistemas que produce. Para ello en sus operaciones internas ha decidido usar los algoritmos de encriptación y dividir su implementación entre los diferentes equipos de desarrollo que la conforman. Según lo observado este proceso se debe llevar a cabo debido a los siguientes componentes que se ha identificado.

Causas:

- Una vulnerabilidad detectada en los sistemas informáticos el cual consiste en un exploit que puede generar exposición de los datos confidenciales de los usuarios registrados en la empresa.
- Firewalls o cortafuegos mal configurados.
- Las vulnerabilidades de los sistemas operativos y la desactualización de los "parches" concernientes a su seguridad.
- La suplantación de identidades.

Síntomas:

- Tiempo de respuesta prolongado en operaciones sencillas.
- Inaccesibilidad al sistema.

Definición:

Construir algoritmos de encriptación que aprovechan las propiedades de los números primos para estos, por ejemplo el algoritmo de RSA (Véase) utiliza números primos para encriptar datos. Todo esto con el fin de aportar en la mejora de la seguridad de los sistemas ofrecidos por la empresa.

Definición de Requerimientos Funcionales:

- ***Requerimiento funcional N# 1***

Nombre	Ingresar el número máximo o tope (n) para la búsqueda de los números primos.
Resumen	Permite al usuario ingresar el valor (sea este primo o no) hasta el que se realizará la búsqueda de los números primos.
Entradas	Un valor que representa el tope máximo para la búsqueda de números primos.
Salida	La lista de números primos encontrados hasta el valor dado.

- ***Requerimiento funcional N#2***

Nombre	Generar una matriz de números (desde 1 hasta n) cuadrada o en su defecto lo más cuadrada posible.
Resumen	Permite generar una matriz con los números primos y no primos en una matriz cuadrada.
Entradas	Ninguna
Salida	Una matriz cuyas casillas corresponden a los números naturales de 1 hasta n

- ***Requerimiento funcional N#3***

Nombre	Mostrar la matriz de números (desde 1 hasta n) cuadrada o en su defecto lo más cuadrada posible.
Resumen	Permite generar gráficamente una matriz con los números primos y no primos en una matriz cuadrada. Los números que no sean primos se mostrarán en un color rojo y los que los sean se mostrará en color verde.
Entradas	Ninguna.
Salida	Una matriz cuyas casillas corresponden a los números naturales de 1 hasta n.

2. RECOPILACIÓN DE LA INFORMACIÓN NECESARIA

Numero: Representación abstracta de una cantidad.

Número primo: Se conoce como número primo a cada número que sólo puede dividirse por 1 y por sí mismo. Por citar un ejemplo: 3 es un número primo, mientras que 6 no lo es ya que 6 / 2 = 3 y 6 / 3 = 2.

Criptografía: Según la UPM, es el arte de escribir con clave secreta o de un modo enigmático, entonces la criptografía es el conjunto de técnicas para proteger t/o salvaguardar algún tipo de data confidencial, para ello generalmente

Cifrado: Es la conversión o “disfraz” de un conjunto sensible de datos, para hacer que no sea fácilmente detectado por terceros, por ejemplo el cifrado César que consiste en mover cada letra por un número previamente dado ejemplo y así entonces la oración “Hola Mundo” sería “Ipmb Nvñep”, Cifrado y encriptación son sinónimos

3. BÚSQUEDA DE SOLUCIONES CREATIVAS

Alternativa 1: Lluvia de ideas
Sean N el conjunto de los números naturales tales que tenemos un valor *k* al cual le hallaremos su maximo comun divisor.

Alternativa 2: Idea formal MATEMÁTICA
Si *p* es un número primo y *a* es un número entero positivo que no tiene factores comunes con *p* (es decir, *a* y *p* son primos relativos), entonces el resto de la división de a^{p-1} entre *p* es 1. En términos de congruencias, este teorema queda así:

Si p es un número primo y a es un número entero positivo primo relativo con p, entonces:

$$a^{p-1} \equiv 1 \mod p$$

Alternativa 3: Idea formal MATEMÁTICA
Sean a y b dos números mayores a 2, a es primo si y sólo si, es divisible por el mismo y por 1, considerando al 1 como un no primo, para número muy grandes, se tiene que a%n-1 sea diferente de 0 si es 0 significa que contiene otro divisor. Por tanto se descarta

Alternativa 4: Criba de Eratóstenes
Se tiene una matriz de tamaño n cuyo elementos están ordenados ascendentemente, posteriormente se eliminan los números múltiplos de los primeros primos, por ejemplo para el 2, se eliminan todos los números pares, y así sucesivamente para el 3 se eliminan todos los números múltiplos de éste (Véase)

Alternativa 5: Lluvia de ideas
Usar las ecuaciones diofánticas..

Alternativa 6: Buscar en Google

Para este método se busca en google la cantidad de primos hasta el cual se desea saber por ejemplo Primeros 10 millones de primos

Alternativa 7:

Usar una computadora cuántica y calcular todos los números primos con el siguiente algoritmo

```
Inicio
Leer(N)
Si ( N < 2 ) {
Escribir ("No válido")
sino
Si ( N = 2 ) {
Escribir ("Es primo")
sino
x  = 2
Hacer
A = N / x
```

Si (¿A es entero?) {
Escribir ("No es primo")
goto salir // ir a etiqueta

sino
x = x + 1
Mientras (x < N)
Escribir("Es primo")
}
}
}
salir ; //etiqueta

Fin

Fuente

4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES

Descarte de ideas:
Las alternativas 5,6,7 son inviables. La 6, es inviable debido a que no se realiza ninguna proceso de pensamiento algorítmico, sino que solo se busca resultados previamente calculados.

Respecto a las ecuaciones diofánticas, aunque son bastante útiles en la búsqueda de valores enteros que cumplan ciertas condiciones para un determinado caso, no conocemos la suficiente teoría práctica y de manejo lo cual resultaría en algoritmos improvisados e ineficientes. Además, el sistema de resolución de la búsqueda de números primos con este método consta de un sistema de 14 ecuaciones que genera un polinomio que los obtiene. (Véase) . Por esto decimos que es un proceso bastante tedioso y complicado.

La 7 tiene un coste bastante alto, debido a que las computadoras cuánticas no son accesibles para todo el mundo, además que desconocemos la manera de cómo se programan este tipo de computadoras.

Posibles Soluciones:

Alternativa 1:
e dice que dos o más números son *primos entre sí*, cuando el único divisor común a todos ellos es la unidad. 16. 14. 25 Y 35 son primos entre sí.

Una regla para conocer si un número es primo es dividir sucesivamente por los primos 2, 3. 5. 7. 11, 13, etc., y si se llega, sin obtener cociente exacto, a un cociente entero menor que el divisor, dicho número es primo.

Por ejemplo, propongámonos averiguar, si el número 419 es primo.

Dividiendo el número anterior sucesivamente por los primos 2, 3, 5, 7, 11, 13, 17 y 19, no hemos obtenido división exacta y el cociente ha sido siempre mayor que el divisor; seguidamente, lo

dividimos por el número primo inmediato superior, 23, y hallamos el cociente entero 18, menor que el divisor, de lo que inferimos que el número 419 es primo.

Alternativa 2:

Si *p* es un número primo y *a* es un número entero positivo que no tiene factores comunes con *p* (es decir, *a* y *p* son primos relativos), entonces el resto de la división de a^{p-1} entre *p* es 1. En términos de congruencias, este teorema queda así:

> Si p es un número primo y a es un número entero positivo primo relativo con p, entonces:
>
> $$a^{p-1} \equiv 1 \mod p$$

En principio, este teorema puede ser muy útil para descartar que un número es primo, ya que para que un cierto número *n* sea primo es necesario que se cumpla el pequeño teorema de Fermat para todo entero positivo *a* menor que el propio *n*.

Veamos un ejemplo. Supongamos que queremos saber si el número 413 es primo o no, y vamos a buscar ayuda en el pequeño teorema de Fermat. Tomamos un entero positivo primo relativo con 413, por ejemplo el 2, y aplicamos el test de Fermat. Si 413 fuera primo, tendríamos que $2^{413-1}=2^{412}$ dejaría de resto 1 al dividirlo entre 413, pero en realidad el resto de esa división es 359. Conclusión: 413 no es un número primo. Os invito a que probéis con otros números y que nos contéis vuestros resultados.

Alternativa 3:

Este método consiste en comprobar desde un número dado, por ejemplo 100 los números primos menores a este para esto se recorre y se evalúa la condición, x(x es un numero cualquiera) módulo numero es igual a 0, de ser asi el numero es primo, en caso contrario se descarta como primo

Por ejemplo

5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN

Dependiendo del tamaño de la entrada, la criba de Eratóstenes es bastante eficiente y además conveniente debido a que esta ya funciona desde una matriz, es por ello que se utilizará como primer algoritmo, y si la entrada es menor a 100 ya que para esa es cuando el algoritmo va perdiendo su eficiencia

El algoritmo se analizará a continuación:

Según algunas webs la complejidad de este algoritmo es O(nlog logn)

La otra forma de calcular los números primos, de mayor tamaño es mediante el teorema de Wilson, el cual nos dice lo siguiente:

> Si p es un número primo, entonces $(p-1)! \equiv -1 \pmod{p}$
>
> John Wilson

Y la última forma es, a priori la más ineficiente para números muy grandes, está en la descrita en el siguiente algoritmo (Véase)

Esta consiste en buscar un múltiplo del número dado, iniciando desde el número inmediatamente anterior, esto se consigue recorriendo todos los numeros del numero dado, es decir:

Dado un número n, n mod n-1 != 0, si esta se cumple para todos los números desde 3 hasta n-1, entonces el número es primo de lo contrario no lo es.

6: PSEUDOCODIGO DE LOS ALGORITMOS

Nombre del metodo	**Complejidad Temporal**	**Complejidad Espacial**
ErastotenesCrib(int n)		1
boolean prime[n+1]	1	n+1
for(i = 2 to root square of n)	$\sqrt{n}$	1
if(prime[i]==false)	$\sqrt{n}$ -1	0
for j = i to n/i	$\sqrt{n}$	1
prime[j+i] = true	$\sqrt{n}$ -1	1
return prime	1	1

anSimpleFormAlgorithm(int n)		1
boolean isPrime	1	1
prime[];	1	n
for i = 0 to n	n+1	1
isPrime = true	n	0
for (j=2 j<square_root(i),j++)	$\sqrt{n}$	1
if(i mod j == 0)	$\sqrt{n}$-1	0
isPrime = false	$\sqrt{n}$-1	0
if(IsPrime)	n-1	0
prime[i]==false;	n-1	n
return prime;	1	1

tourAlgorithm(int n)		1
boolean prime[];	1	1
for i = 0 to n	n+1	1
if(isPrime(i))	n^2	0
prime[i] == false;	n-1	0
return prime	1	1

isPrime(int n)		1
int i = n-1	1	1
boolean esPrimo = true	1	1
if n> 2	1	1
while i >1 && esPrimo	n-1	1
if n mod i != 0	n-1	0
i -=1	n-1	0
else	n-1	
esPrimo = false;	n-1	0
else	n-1	
esPrimo = false	n-1	0
return esPrimo:	1	1

7. DISEÑO DE PRUEBAS UNITARIAS

Prueba : El método es capaz de encontrar los números desde 2 hasta n.

Clase	**Método**	**Escenario**	**Resultado**
GenerateNumbers	erastotenesCribAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 10
GenerateNumbers	erastotenesCribAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 100
GenerateNumbers	erastotenesCribAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 1.000

Prueba : El método es capaz de encontrar los números desde 2 hasta n.

Clase	**Método**	**Escenario**	**Resultado**
GenerateNumbers	tourAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 100
GenerateNumbers	tourAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 1.000
GenerateNumbers	tourAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 10.000

Prueba : El método es capaz de encontrar los números desde 2 hasta n.

Clase	**Método**	**Escenario**	**Resultado**
GenerateNumbers	aSimpleFormAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 100
GenerateNumbers	aSimpleFormAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los números primos	Lista de los números primos desde 2 hasta 1.000
GenerateNumbers	aSimpleFormAlgorithm(n)	Se crea una nueva instancia de la clase con un ArrayList vacío en el cual se almacenarán los	Lista de los números primos desde 2 hasta 10.000

		números primos	