Universidad de Alcalá

DEPARTAMENTO DE AUTOMÁTICA
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Máster en Ciberseguridad
CRIPTOGRAFÍA APLICADA

Implementación del algoritmo *paso gigante-paso enano* para el cómputo de logaritmos discretos

OBJETIVO

En esta práctica se pretende implementar mediante lenguaje de programación Java el algoritmo conocido como *paso gigante-paso enano*[1] para el cálculo de logaritmos discretos.

ENTREGABLE

Se espera que, como resultado de la ejecución de la práctica, el alumno entregue el código fuente preparado para ser compilado y ejecutado. Además, deberá redactar un informe describiendo los pasos realizados para la implementación y comentando los resultados obtenidos. Todo ello deberá ser entregado en la plataforma dentro del plazo marcado.

ASPECTOS TEÓRICOS

Supongamos un grupo (multiplicativo) G, de orden m, y un generador del grupo, g. Por definición de generador, se verifica que cualquier elemento del grupo se puede obtener por medio de alguna potencia de g. Dicho formalmente, para cualquier $h \in G$ se tiene que

$$h = g^{\alpha},$$

donde $\alpha \in \mathbb{Z}$ es lo que llamamos el *logaritmo discreto* de h en base g, esto es

$$\alpha = \log_g h.$$

Obtener el logaritmo discreto no es fácil desde un punto de vista computacional, pues los algoritmos (clásicos) más eficientes que conocemos necesitan un tiempo (sub)-exponencial. En esta práctica se trata de implementar uno de ellos conocido con el nombre de algoritmo de *paso gigante-paso enano*, cuyo tiempo esperado de ejecución es

$$\mathcal{O}\left(2^{\frac{1}{2}\log m} \cdot \mathrm{pol}(\log m)\right),$$

donde m es, como se ha dicho, el orden del grupo G.

Descripción del algoritmo

Dado el grupo G, de orden m y el generador g, se nos da un $y \in G$, y se nos pide calcular su logaritmo, β, es decir, aquel valor tal que $y = g^{\beta}$.

El algoritmo se desarrolla en dos fases, de donde el algoritmo recibe su nombre. La primera parte es el *paso de gigante*:

[1] Conocido en inglés como *baby step-giant step*.

- Se calcula t como la raíz cuadrada entera de m.
- Se calculan y almacenan las parejas (j, g^{jt}), para $0 \le j \le t$, ordenadas por la segunda componente.

La segunda parte es el *paso de enano*:

- Desde $i = 0$, se calcula ahora $y \cdot g^i = g^{\beta+i}$ y se busca en la segunda componente de la tabla anterior, incrementando el valor de i si no se encuentra.
- Antes o después, algún valor de i, digamos i^*, dará coincidencia con alguna entrada, digamos j^*, es decir
$$y \cdot g^{i^*} = g^{\beta+i^*} = g^{j^* t},$$
y, despejando en los exponentes el valor de β, obtenemos finalmente el logaritmo buscado como
$$\beta = j^* t - i^* \pmod{m}.$$

ENTRADA:	Un primo p, un generador g en $\mathbb{Z}_p^*$, y un $y \in \mathbb{Z}_p^*$
SALIDA:	El logaritmo discreto de y en base g

```
01: PGPE(p,g,y) {
02:
03:   t = ⌊√(p−1)⌋
04:
05:   // Paso de gigante
06:   s = InitKeyHash()
07:   for j ∈ {0, ..., t} {
08:     v = g^(jt)
09:     InsertKeyValue(s,v,j)
10:   }
11:
12:   // Paso de enano
13:   for i ∈ {0, ..., t} {
14:     v = y · g^i
15:     j = SearchKey(s,v)
16:     if (j <> NO-ENCONTRADO)
17:       return jt − i (mód p − 1)
18:   }
19:   return NO-ENCONTRADO
20: }
```

Figura 1: Algoritmo *paso gigante-paso enano*

DESARROLLO DE LA PRÁCTICA

Utilizando el entorno NetBeans se trata de implementar el algoritmo *paso gigante-paso enano*.

El grupo que se debe utilizar es $\mathbb{Z}_p^*$ multiplicativamente, junto con un generador de él, cuyo orden es, obviamente, $m = p - 1$. Es importante recordar que las operaciones en el grupo son todas ellas módulo p.

Además, es también necesario usar la biblioteca de números grandes `BigInteger` (u otra equivalente), para poder manejar y operar con números de tamaño arbitrariamente grande. En Java puede importarse la biblioteca `BigInteger` mediante la orden:

```
import java.math.BigInteger;
```

Los pasos del algoritmo descrito más arriba están resumidos en "pseudo-código" en la figura 1.

APLICACIÓN

Como aplicación del algoritmo, se pide utilizarlo para encontrar el logaritmo discreto de cada y en sus respectivos grupos $\mathbb{Z}_p^*$, con generador g:

Caso	p del grupo $\mathbb{Z}_p^*$	g del grupo	y
1	65537	3	27861
2	1073741827	2	503593909
3	4294967311	3	3333860011
4	1099511627791	3	73096380924
5	35184372088891	3	24056225665222
6	1125899906842679	11	12761818229206

Se pide representar gráficamente el tiempo necesario para encontrar cada logaritmo discreto en función del número de bits de cada candidato. A la vista de la gráfica, ¿nos encontramos ante un algoritmo de tiempo polinómico?

Nota: En el caso 6 el programa podría quedarse "atascado" por la gran cantidad de memoria necesaria para almacenar la tabla.