

# Teoria de Números Computacional 21/22

## Trabalho Prático 2

Grupo:

- Ivo Miguel Gomes Lima (A90214)
- Tiago dos Santos Silva Peixoto Carriço (A91695)

## Contextualização

Para o segundo trabalho prático foi-nos pedido a implementação e explicação do [Algoritmo de Shanks](https://en.wikipedia.org/wiki/Baby-step_giant-step) ([https://en.wikipedia.org/wiki/Baby-step\\_giant-step](https://en.wikipedia.org/wiki/Baby-step_giant-step)), baby-step giant-step, que permite resolver o Problema do Logaritmo Discreto, tendo em vista o caso de uma raiz primitiva  $r$  de  $\mathbb{Z}_p^*$ .

Tal como no primeiro trabalho prático foi necessário o uso do [SageMath](https://www.sagemath.org) (<https://www.sagemath.org>), e a consulta da secção 3.6.2 (Cap 3) do *Handbook of Applied Cryptography*, A. Menezes, P. van Oorschot, S. Vanstone, CRC Press, 1996 disponibilizado pelo docente da cadeira.

## Problema do Logaritmo Discreto

Existem muitos sistemas de criptografia cuja segurança é baseada na dificuldade em resolver logaritmos discretos. Algebricamente, o logaritmo é um expoente. Mais precisamente, se  $1 \neq \alpha > 0$  é um número real, então para valores positivos de  $\beta \in \mathbb{R}$ , o logaritmo de  $\beta$  na base  $\alpha$  deve ser elevado para produzir  $\beta$ .

Neste trabalho abordaremos um algoritmo para grupos arbitrários, isto é, aqueles que não exploram qualquer propriedade específica do grupo. Para tal apresentaremos um algoritmo característico denominado Algoritmo de *Shanks*.

## Criação do Algoritmo de Shanks

[Daniel Shanks](https://en.wikipedia.org/wiki/Daniel_Shanks) ([https://en.wikipedia.org/wiki/Daniel\\_Shanks](https://en.wikipedia.org/wiki/Daniel_Shanks)), desenvolveu em 1973 um método eficiente para calcular logaritmos discretos. Para este algoritmo necessitamos de enumerar os elementos de  $G$ .

Tomando  $G$  como grupo e  $\alpha \in G$  um elemento de ordem finita (gerador). Dado  $\beta \in \langle \alpha \rangle$ , existe um único natural  $x$ ,  $0 \leq x \leq |\langle \alpha \rangle| - 1$  tal que  $\beta = \alpha^x$ , portanto, o logaritmo discreto de  $\beta$  na base  $\alpha$  é bem definido.

Seja  $n \in \mathbb{N}$ ,  $n \geq |\langle \alpha \rangle|$  e  $m = \lceil \sqrt{n} \rceil$ . Dado  $\beta \in G$ , vamos calcular o logaritmo discreto isto é  $x = \log_\alpha \beta$ .

Tomando as hipóteses acima, concluímos que para a elaboração do código devemos:

Calcular o  $m = \lceil \sqrt{n} \rceil$ , sendo  $n$  a ordem do grupo. De seguida fazemos a construção de pares  $(j, \alpha^j)$ , com  $0 \leq j < m$  que serão inseridos numa tabela de *Hash* por forma a otimizar a pesquisa. Após essa construção computamos  $\alpha^{-m}$ .

Por fim procuramos entre os pares aquele em que a segunda coordenada é igual a  $(\beta \times (\alpha^{-m})^i) \mod p$ , onde  $p$  é que caso apareça implica o calculo de  $i \times m + j$ , que é a solução de  $x \equiv \log_\alpha \beta \mod n$ .

Podemos então concluir que o Algoritmo de *Shanks* é determinístico, tendo um tempo de execução de  $O(\sqrt{n})$ .

```
In [1]: def shanks(a, b, n, p):
        Zn = IntegerModRing(p)
        m = ceil(sqrt(n))

        tabela = {}
        j = 0
        while j < m:
            tabela[Zn(a ^ j)] = j
            j += 1

        a_e_m = Zn(a ^ -m)

        for i in range(m):
            y = Zn(b * a_e_m^i)
            if y in tabela:
                j = tabela[y]
                return i * m + j

        return "Não foi encontrada solução."
```

## Exemplos

### Exemplo 1

Imaginemos que queremos aplicar o Algoritmo de *baby – step giant – step* em  $\mathbb{Z}_{113}^*$  teremos então que  $p = 113$ , sendo  $\alpha = 3$  um gerador do grupo cíclico  $G$  que possui uma ordem  $n = 112$ . Considerando  $\beta = 57$  significa que podemos determinar que o valor de  $x \equiv \log_3 57 \mod 112$ , que será:

```
In [2]: a = 3
        b = 57
        p = 113

        Zn = IntegerModRing(p)
        n = Zn(a).multiplicative_order()
        n
```

Out[2]: 112

```
In [3]: x = shanks(a, b, n, p)
        x
```

Out[3]: 100

```
In [4]: a^x % p == b
```

Out[4]: True

## Exemplo 2

Neste segundo exemplo aplicamos o Algoritmo de *Shanks* com um  $p = 53$ , um gerador  $\alpha = 2$ , ordem  $n = 52$  e que tomando um  $\beta = 45$  fará o valor de  $x \equiv \log_2 45 \pmod{52}$ , ser:

```
In [5]: a = 2
        b = 45
        p = 53

        Zn = IntegerModRing(p)
        n = Zn(a).multiplicative_order()
        n
```

Out[5]: 52

```
In [6]: x = shanks(a, b, n, p)
        x
```

Out[6]: 29

```
In [7]: a^x % p == b
```

Out[7]: True

## Exemplo 3

Para este último exemplo queremos usar o Algoritmo em  $\mathbb{Z}_{53}^*$ , isto é,  $p = 53$ , com um gerador  $\alpha = 18$  e ordem  $n = 52$ . O  $\beta = 12$  e queremos calcular o valor de  $x \equiv \log_{18} 12 \pmod{52}$ , que é:

```
In [8]: a = 18
        b = 12
        p = 53

        Zn = IntegerModRing(p)
        n = Zn(a).multiplicative_order()
        n
```

Out[8]: 52

```
In [9]: x = shanks(a, b, n, p)
        x
```

Out[9]: 5

```
In [10]: a^x % p == b
```

Out[10]: True