

3. Os inteiros. Criptografia

Se eu escrever

$$11 + 22 = 9$$

dirão *ele não sabe somar!*

Por outro lado, se eu disser *são 11 horas, daqui a 22 horas serão 33 horas* dirão *e além disso, não sabe que um dia tem só 24 horas!* São 11 horas, dentro de 22 horas serão $33 - 24 = 9$ horas.

Temos então que decidir,

$$11 + 22 = 33 \quad \text{ou} \quad 11 + 22 = 9?$$

Bem, na aritmética usual

$$11 + 22 = 33$$

mas quando calculamos as horas, $11 + 22 = 9$. Portanto, a aritmética que usamos para calcular as horas é uma aritmética um pouco diferente da habitual, na qual 24 conta como zero, isto é, $24 = 0$. Esta aritmética chama-se *aritmética módulo 24*. Para a distinguir da aritmética habitual escrevemos

$$11 +_{24} 22 =_{24} 9.$$

Note que isto é verdade porque $11 + 22 = 33 = 24 + 9$.

Quanto à multiplicação, $11 \times_{24} 22 =_{24} 2$ pois $11 \times 22 = 242 = (10 \times 24) + 2$. Para obter $242 = (10 \times 24) + 2$ basta fazer a divisão de 242 por 24:

$$\begin{array}{r} 242 \\ 002 \\ \underline{02} \end{array} \quad \begin{array}{r} \underline{24} \\ 10 \end{array} \qquad \begin{array}{r} 242 \\ \underline{240} \\ 2 \end{array} \quad \begin{array}{r} \underline{24} \\ 10 \end{array}$$

O quociente é 10 e o resto é 2.

Definição. Dados dois inteiros m e n , diz-se que r é o *resto* da divisão inteira de n por m , e denota-se por $r = n \bmod m$, se $0 \leq r < |m|$ e $n = q \times m + r$ para algum inteiro q . No caso particular em que $r = 0$, diz-se que m *divide* n e escreve-se $m \mid n$.

No Maple esta relação pode ser implementada da seguinte forma, com a ajuda da package `numtheory` de teoria dos números:

```
> with(numtheory);
> 23 mod 7;
      2
> 23 mod (-7);
      2
> -23 mod 7;
```

5

```
> divisor := (m,n) -> evalb(n mod m=0);
> divisor(2,6);
```

true

```
> divisor(2,7);
```

false

Portanto

- $a +_{24} b = (a + b) \bmod 24$.
- $a \times_{24} b = (a \times b) \bmod 24$.

Naturalmente, o número 24 não tem nada de particular. Podemos considerar a aritmética módulo 43, onde $43 = 0$, ou a aritmética módulo 10, onde $10 = 0$; ou naturalmente a aritmética módulo um número n muito grande, por exemplo

$$n = 3469016345521790021102382940567489953,$$

a aritmética onde este número n é igual a zero.

Assim, podem-se definir operações $+_n$ e \times_n sobre o conjunto $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$:

$$a +_n b = (a + b) \bmod n, \quad a \times_n b = (a \times b) \bmod n.$$

É esta a chamada *aritmética modular*¹⁴ nos inteiros.

Na aritmética modular todas as propriedades usuais da soma e da multiplicação dos números inteiros continuam válidas. Por exemplo

$$\begin{aligned} (a +_n b) +_n c &= a +_n (b +_n c) \\ (a \times_n b) \times_n c &= a \times_n (b \times_n c) \\ a \times_n (b +_n c) &= (a \times_n b) +_n (a \times_n c). \end{aligned}$$

Mas a particularidade da aritmética módulo n é que algumas vezes um inteiro a pode ter um inverso $\frac{1}{a}$ que é ainda um inteiro! Isto é, para um determinado número a módulo n , pode existir um número b módulo n tal que

$$a \times_n b = 1.$$

Por exemplo, para $n = 10$, como $7 \times 3 = 21 = (2 \times 10) + 1$, então $7 \times_{10} 3 = 1$.

Dados inteiros a e m , a notação

$$a \bmod m$$

¹⁴Para mais informação e manipulação destas operações no caso $1 \leq n \leq 10$ vá a www.atractor.pt/mat/alg_controlo/arit_modular/mod_texto.htm.

representa o resto da divisão inteira de a por m . Por exemplo,

$$10 \bmod 5 = 0, \quad 7 \bmod 5 = 2 \quad \text{e} \quad 2 \bmod 5 = 2.$$

Isto permite definir a chamada *relação de congruência módulo n* entre inteiros:

Definição. Dados inteiros a e b e um natural n , diz-se que a é *congruente com b módulo n* , e escreve-se $a \equiv_n b$, se $a \bmod n = b \bmod n$, isto é, $(a - b) \bmod n = 0$ (ou seja, quando $a - b$ é múltiplo de n).

Verifique que esta relação satisfaz as seguintes propriedades:

(C1) Trata-se de uma relação de equivalência, isto é, para quaisquer a, b, c, d em \mathbb{Z} :

- $a \equiv_n a$ (reflexiva).
- Se $a \equiv_n b$ então $b \equiv_n a$ (simétrica).
- Se $a \equiv_n b$ e $b \equiv_n c$ então $a \equiv_n c$ (transitiva).

(C2) Se $a \equiv_n b$ e $c \equiv_n d$ então $a + c \equiv_n b + d$.

(C3) Se $a \equiv_n b$ e $c \equiv_n d$ então $a \times c \equiv_n b \times d$.

Aplicações.

(1) Códigos. Com esta aritmética temos já uma maneira simples de codificar e decodificar uma informação! Efectivamente, podemos multiplicar por 7 módulo 10 para codificar a informação e depois, multiplicar por 3 módulo 10 para decodificar. Fazer estas duas operações consecutivamente corresponde exactamente a multiplicar por 1 módulo 10, isto é, não fazer nada! Recuperamos assim a informação inicial!

Consideremos um exemplo. Um cartão de multibanco tem um código secreto, em geral um número de quatro algarismos. Claro que não é prudente escrevermos este código sobre o cartão. Mas porque não? Por exemplo, para o código 7938, posso multiplicar cada algarismo por 7 módulo 10

$$7 \times_{10} 7 = 9, \quad 9 \times_{10} 7 = 3, \quad 3 \times_{10} 7 = 1, \quad 8 \times_{10} 7 = 6$$

e escrever os resultados sobre o cartão: 9316. Para recuperar o código correcto, basta multiplicar por 3 módulo 10:

$$9 \times_{10} 3 = 7, \quad 3 \times_{10} 3 = 9, \quad 1 \times_{10} 3 = 3, \quad 6 \times_{10} 3 = 8.$$

(2) Números aleatórios. Usámos na secção 1.3 a função `randperm` do `Maple` que permite gerar uma permutação aleatória. É também possível gerar números aleatórios com as instruções `rand()` (gera aleatoriamente um número natural com 12 algarismos) e `rand(n)` (gera aleatoriamente um número natural entre 1 e $n-1$). A geração destes números é muito útil em simulações computacionais. Diversos métodos têm sido criados para gerar números destes. Em rigor, nenhum destes métodos gera números perfeitamente aleatórios, por isso é habitual chamar-lhes *números pseudo-aleatórios*.

O método mais comum é o chamado *método das congruências lineares*. Escolhemos quatro inteiros: o módulo m , o multiplicador a , o incremento c e a raiz x_0 , com $2 \leq a < m$, $0 \leq c < m$ e $0 \leq x_0 < m$. Gera-se uma sequência de números pseudo-aleatórios $\{x_n\}$, com $0 \leq x_n < m$ para qualquer n , usando sucessivamente a fórmula

$$x_{n+1} = (ax_n + c) \bmod m.$$

Por exemplo, a sequência de números pseudo-aleatórios gerada escolhendo $m = 9$, $a = 7$, $c = 4$ e $x_0 = 3$ é a seguinte:

$$\begin{aligned} x_1 &= (7x_0 + 4) \bmod 9 = 25 \bmod 9 = 7 \\ x_2 &= (7x_1 + 4) \bmod 9 = 53 \bmod 9 = 8 \\ x_3 &= (7x_2 + 4) \bmod 9 = 60 \bmod 9 = 6 \\ x_4 &= (7x_3 + 4) \bmod 9 = 46 \bmod 9 = 1 \\ x_5 &= (7x_4 + 4) \bmod 9 = 11 \bmod 9 = 2 \\ x_6 &= (7x_5 + 4) \bmod 9 = 18 \bmod 9 = 0 \\ x_7 &= (7x_6 + 4) \bmod 9 = 4 \bmod 9 = 4 \\ x_8 &= (7x_7 + 4) \bmod 9 = 32 \bmod 9 = 5 \\ x_9 &= (7x_8 + 4) \bmod 9 = 39 \bmod 9 = 3 \end{aligned}$$

Como $x_9 = x_0$ e cada termo na sequência só depende do anterior, a sequência terá nove números diferentes antes de se começar a repetir:

$$3, 7, 8, 6, 1, 2, 0, 4, 5, 3, 7, 8, 6, 1, 2, 0, 4, 5, 3, \dots$$

A maioria dos computadores usa este método para gerar números pseudo-aleatórios. Por exemplo, é muito utilizado o sistema módulo $m = 2^{31} - 1$ com incremento $c = 0$ e multiplicador $a = 7^5 = 16\,807$, que permite gerar $2^{31} - 2$ números antes que a repetição comece.

(3) Cálculo do máximo divisor comum. O algoritmo mais antigo que se conhece, e que aparece no livro VII dos *Elementos* de Euclides (c. 325 a.C. - 265 a.C.), calcula o *máximo divisor comum* $\text{mdc}(a, b)$ de dois inteiros a e b .

Exemplo. Consideremos os inteiros $a = 252$ e $b = 54$. Dividindo a por b obtemos $a = 252 = 54 \times 4 + 36$. Tornando a dividir, agora b pelo resto 36, $54 = 36 \times 1 + 18$. Continuando o processo, chegamos a uma divisão exacta (porquê?), e o processo pára: $36 = 18 \times 2 + 0$. É fácil ver que 18, o último resto não nulo, é o máximo divisor comum de a e b .

$$\begin{aligned} 252 &= 54 \times 4 + 36 \\ 54 &= 36 \times 1 + \boxed{18} \\ 36 &= 18 \times 2 + \boxed{0} \end{aligned}$$

$$\therefore \text{mdc}(252, 54) = 18$$

Não se trata de uma coincidência: não é difícil provar que, seguindo este procedimento para quaisquer outro par de inteiros positivos, o último resto não nulo é sempre igual a $\text{mdc}(a, b)$. Este é o algoritmo de Euclides:

```

procedure mdc(a, b : inteiros positivos)
  x := a
  y := b
  while y ≠ 0
  begin
    r := x mod y
    x := y
    y := r
  end {x é o mdc(a, b)}

```

O algoritmo de Euclides é um dos resultados básicos dos números inteiros.

(4) Criptografia¹⁵: cifra de César. As congruências utilizam-se muito na criptografia. O exemplo mais simples (e muito antigo, remonta a Júlio César) é a chamada *cifra de César*. Ele usava um método de escrita de mensagens secretas transladando cada letra do alfabeto para três casas mais à frente:

A	B	C	D	E	F	G	H	I	J	L	M	N	O	P	Q	R	S	T	U	V	X	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
D	E	F	G	H	I	J	L	M	N	O	P	Q	R	S	T	U	V	X	Z	A	B	C

Este sistema de encriptação pode ser descrito matematicamente de forma muito abreviada: substituímos cada letra por um inteiro de 0 até 22, baseado na sua posição no alfabeto:

A	B	C	D	E	F	G	H	I	J	L	M
0	1	2	3	4	5	6	7	8	9	10	11
N	O	P	Q	R	S	T	U	V	X	Z	
12	13	14	15	16	17	18	19	20	21	22	

Portanto o método de César é definido pela função f que aplica cada inteiro n , $0 \leq n \leq 22$, no inteiro

$$f(n) = (n + 3) \bmod 23.$$

Por exemplo,

$$\begin{array}{c}
 X \longleftrightarrow 21 \\
 \downarrow \\
 f(21) = 24 \bmod 23 = 1 \longleftrightarrow B.
 \end{array}$$

Teste. Como fica a mensagem “DESCOBRI A SOLUCAO” depois de encriptada pela cifra de César ?

¹⁵A *criptografia* é a parte da *criptologia* (ciência dos códigos) que se dedica ao estudo de mensagens secretas e dos processos de *encriptação*, ou seja, de escrita de mensagens secretas.

Para recuperar a mensagem original a partir da mensagem encriptada basta considerar a função inversa f^{-1} que transforma um inteiro n , $0 \leq n \leq 22$, em $f^{-1}(n) = (n - 3) \bmod 23$.

Podemos generalizar a cifra de César trasladando b casas em vez de três:

$$f(n) = (n + b) \bmod 23.$$

É claro que a cifra de César é um método de encriptação muito pouco seguro. Podemos melhorá-lo um pouco definindo, mais geralmente, $f(n) = (an + b) \bmod 23$, com a e b inteiros escolhidos de modo a garantir que f é uma bijecção.

Teste. Que letra substitui J com a função encriptadora $f(n) = (7n + 3) \bmod 23$?

Teste. Descodifique a mensagem

PIBE ◊ D ◊ W@P

(onde o símbolo ◊ indica um espaço em branco) que foi encriptada utilizando o alfabeto da figura seguinte e a função $f(p) = (22p + 25) \bmod 29$.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z	W	*	@	◊
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Números primos.

Quais são os elementos de \mathbb{Z}_n , chamados *invertíveis*, que têm inverso relativamente à operação \times_n ? Estes elementos estão intimamente ligados aos números primos.

Definição. Um natural $p \geq 2$ diz-se *primo* quando, para qualquer natural n , se n divide p então $n = 1$ ou $n = p$.

No Maple o predicado `isprime` permite testar se um número é primo¹⁶ e a função `ithprime` lista o i -ésimo número primo:

```
> isprime(23);
      true

> isprime(1);
      false

> ithprime(1); # o primeiro numero primo
      2
```

¹⁶Testar se um número é primo é um problema de *complexidade polinomial*. Este resultado só foi provado muito recentemente (em 2002) por uma equipa de investigadores do *Indian Institute of Technology*, depois de muitas tentativas goradas durante o século passado.

```
> ithprime(30000); # o 30000-esimo numero primo
```

```
350377
```

```
> seq(ithprime(i), i=1..100); # Lista dos primeiros 100 primos
```

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541
```

Os números primos têm um papel fundamental relativamente aos outros números naturais por causa do seguinte resultado:

Teorema Fundamental da Aritmética. *Todo o natural $n \geq 2$ pode escrever-se de maneira única, a menos da ordem dos factores, da seguinte forma:*

$$n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

onde cada p_i é primo e $e_i > 0$.

No Maple podemos obter esta decomposição em primos através da função `ifactor`:

```
> ifactor(6600);
```

```
(2)3 (3) (5)2 (11)
```

```
> ifactor(27236209);
```

```
(7)2 (11) (13)3 (23)
```

Ao contrário do que acontece com o problema da verificação da primalidade de um número, não se conhece nenhum algoritmo que, em tempo polinomial, consiga realizar esta decomposição em primos. Mais, conjectura-se que um tal algoritmo não existe.

Definição. Dois naturais m e n dizem-se *primos entre si* se $\text{mdc}(m, n) = 1$ (diz-se também que m é *coprimo* de n).

Podemos agora caracterizar facilmente os elementos de \mathbb{Z}_n que são invertíveis para a multiplicação \times_n :

Proposição. *Um natural $a \in \mathbb{Z}_n$ é invertível se e só se a e n são primos entre si.*

Prova. “ \Rightarrow ”: $ab \equiv_n 1$ significa que $ab = nq + 1$, isto é, $ab - nq = 1$, para algum inteiro q . Então, se d é um divisor comum de a e n , será um divisor de $ab - nq = 1$, pelo que necessariamente $d = 1$ ou $d = -1$. Isto mostra que $\text{mdc}(a, n) = 1$.

“ \Leftarrow ”: Se $\text{mdc}(a, n) = 1$ então, pelo algoritmo de Euclides (usado em ordem inversa — ver exemplo na página seguinte), existem inteiros s e t tais que $1 = sa + tn$, ou seja, $sa = n \times (-t) + 1$,

o que mostra que $sa \equiv_n 1$. Consequentemente, todos os números da forma $s + kn$ ($k \in \mathbb{Z}$) são solução da equação $xa \equiv_n 1$ e um deles pertence necessariamente a \mathbb{Z}_n . Esse será o inverso de a em \mathbb{Z}_n . \square

Portanto, em $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ todos os elementos não nulos são invertíveis:

$$1 \times_5 1 = 1, \quad 2 \times_5 3 = 1, \quad 3 \times_5 2 = 1, \quad 4 \times_5 4 = 1.$$

Mais geralmente, se n é primo todos os elementos não nulos de \mathbb{Z}_n são invertíveis. Em $\mathbb{Z}_4 = \{0, 1, 2, 3\}$ só o 1 e o 3 são invertíveis; em $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ só o 1 e o 5 são invertíveis. Em \mathbb{Z}_{10} , como vimos no início da seção, o 7 é invertível.

Para cada n , o número de coprimos de n menores ou iguais a n é dado pela *função de Euler* (ou função *totiente*)

$$\begin{aligned} \phi : \mathbb{N} &\rightarrow \mathbb{N} \\ n &\mapsto \#\{m : m \leq n \text{ e } m \text{ é coprimo de } n\} \end{aligned}$$

cujos valores podemos calcular com o **Maple**, recorrendo à *package numtheory*:

```
> with(numtheory);
> phi(4); phi(5); phi(6); phi(10);
```

```
2
4
2
4
```

```
> phi(107); phi(106);
```

```
106
52
```

Portanto, em \mathbb{Z}_{107} há 106 elementos invertíveis e em \mathbb{Z}_{106} só há 52.

Acabámos de obter uma solução para as congruências do tipo $ax \equiv_n 1$. Mais geralmente:

Resolvendo uma congruência. Sejam a, b, n inteiros. Se $\text{mdc}(a, n) = 1$ então a equação $ax \equiv_n b$ tem uma solução:

- (1) Determine inteiros s e t tais que $1 = as + nt$
(usando o algoritmo de Euclides por ordem inversa)
- (2) Então $x = bs$ é uma solução da congruência
(o conjunto completo de soluções é $\{bs + nk \mid k \in \mathbb{Z}\}$)

Teste. Resolva a equação $10x \equiv_{27} 5$.

Solução. Usemos o algoritmo de Euclides para determinar $\text{mdc}(10, 27)$:

$$\begin{aligned} 27 &= 10 \times 2 + 7 \\ 10 &= 7 \times 1 + 3 \\ 7 &= 3 \times 2 + 1 \end{aligned}$$

Portanto $\text{mdc}(10, 27) = 1$. Agora invertamos o processo para encontrar inteiros s e t tais que $1 = 10s + 27t$:

$$\begin{aligned} 1 &= 7 - 3 \times 2 \\ &= 7 - (10 - 7 \times 1) \times 2 \\ &= 7 \times 3 - 10 \times 2 \\ &= (27 - 10 \times 2) \times 3 - 10 \times 2 \\ &= 10 \times (-8) + 27 \times 3. \end{aligned}$$

Portanto $s = -8$ e $t = 3$. Logo $x = bs = -40$ é uma solução, e o conjunto completo de soluções é $\{-40 + 27k \mid k \in \mathbb{Z}\}$.

Teorema de Fermat-Euler.¹⁷

- (a) Se p é um primo que não divide x então $x^{p-1} \equiv_p 1$.
 (b) Se p e q são primos que não dividem x então $x^{(p-1)(q-1)} \equiv_{pq} 1$.

Prova. Provamos somente (a). Para isso consideremos os inteiros

$$x \bmod p, \quad 2x \bmod p, \quad \dots, \quad (p-1)x \bmod p. \quad (*)$$

Estes números são todos distintos, dois a dois:

Se $nx \bmod p = mx \bmod p$, com n, m entre 1 e $p-1$, então $nx \equiv_p mx$, isto é, $nx - mx \equiv_p 0$, ou seja, p divide $(n-m)x$. Mas p é primo e não divide x logo terá que dividir o outro factor $n-m$. Como $n-m$ é um número entre 0 e $p-1$, o único número destes que é múltiplo de p é o zero. Portanto, $n-m=0$, ou seja, $n=m$.

Assim, como todos estes $p-1$ números são distintos e pertencem ao conjunto $\{1, 2, \dots, p-1\}$, a lista (*) constitui um rearranjo (permutação) dos números $1, 2, \dots, p-1$. Portanto, o produto dos números da lista é igual ao produto dos números $1, 2, \dots, p-1$:

$$x^{p-1}(p-1)! \bmod p = (p-1)! \bmod p.$$

Daqui decorre que $x^{p-1}(p-1)! \equiv_p (p-1)! \Leftrightarrow (p-1)!(x^{p-1} - 1) \equiv_p 0$. Portanto p divide $(p-1)!(x^{p-1} - 1)$. Como p não pode dividir $(p-1)!$, terá então que dividir $x^{p-1} - 1$, o que significa que $x^{p-1} \equiv_p 1$. \square

¹⁷Este teorema (proposição (a)) foi obtido pelo matemático francês Fermat por volta de 1630, e generalizado pelo matemático suíço Euler um século mais tarde (proposição (b)).

Mais uma aplicação à Criptografia: o sistema RSA de chave pública.

Os resultados sobre os inteiros que acabámos de estudar estão na base de toda a criptografia actual, que permite a troca de mensagens confidenciais por intermédio de um canal público, supondo que os agentes comunicantes, digamos a Alice e o Bruno, não partilham segredo nenhum.

Se a Alice quiser enviar uma mensagem x ao Bruno, pede-lhe para ele gerar um par de chaves, uma chave pública u (conhecida por toda a gente) e uma chave privada v (conhecida apenas pelo Bruno). As chaves u e v são aplicações do espaço das mensagens para o espaço das mensagens e, para que o sistema funcione bem e permita manter o secretismo na comunicação, devem ter as seguintes propriedades:

(P1) $v(u(x)) = x$ para qualquer mensagem x .

(P2) deve ser difícil obter x conhecendo $u(x)$ e não conhecendo v .

O protocolo funciona do seguinte modo:

- (1) A Alice envia a mensagem $u(x)$ ao Bruno pelo canal público.
- (2) O Bruno recupera a mensagem original x aplicando v a $u(x)$.

Ao definirmos um *sistema criptográfico* deveremos explicitar o espaço das mensagens bem como as aplicações u e v . Um dos sistemas mais utilizados hoje em dia é o *sistema RSA* cujo nome deriva dos seus criadores (Rivest, Shamir e Adleman, em 1976).

A família de sistemas criptográficos RSA é definida do seguinte modo:

- espaço de mensagens: $\mathbb{Z}_n = \{0, 1, 2, \dots, n-1\}$, onde $n = p \times q$ para algum par de primos p, q ,
- $u(x) = x^a \pmod n$, para qualquer $x \in \mathbb{Z}_n$,
- $v(y) = y^b \pmod n$, para qualquer $y \in \mathbb{Z}_n$,

onde a e b são tais que $a \times b \pmod{(p-1)(q-1)} = 1$.

Será de facto um bom sistema, isto é, satisfaz as propriedades (P1) e (P2)?

Os conceitos e resultados relativos aos números inteiros apresentados nesta secção permitem-nos verificar a propriedade (P1), como veremos mais adiante. Quanto à propriedade (P2), a sua confirmação é ainda hoje um problema em aberto¹⁸.

Este sistema permite enviar mensagens encriptadas por uma chave pública a , mas para desencriptar a mensagem o receptor precisa de ter uma chave privada b (só do seu conhecimento).

Sejam

$$p \text{ e } q \text{ primos, } n = pq, \quad m = (p-1)(q-1).$$

¹⁸Trata-se de um dos problemas em aberto mais importantes da matemática, com grandes implicações práticas: até hoje, ninguém conseguiu demonstrar se o sistema RSA verifica a propriedade (P2), apesar de todos os especialistas conjecturarem que isso seja verdadeiro. Portanto, toda a criptografia actual assenta, não numa certeza absoluta, mas numa conjectura.

Consideremos ainda a tal que $\text{mdc}(a, m) = 1$ e seja b a solução da congruência $ab \equiv_m 1$.

No sistema RSA, podemos começar por traduzir as mensagens (sequências de letras) em sequências de inteiros (como na cifra de César):

A	B	C	D	E	F	G	H	I	J	L	M
00	01	02	03	04	05	06	07	08	09	10	11
N	O	P	Q	R	S	T	U	V	X	Z	
12	13	14	15	16	17	18	19	20	21	22	

O inteiro x daí resultante é depois transformado, com a ajuda da chave pública a , num inteiro

$$u(x) = x^a \pmod{n}.$$

Teste. Codifique a mensagem HELP usando o sistema RSA com $p = 43$, $q = 59$ e $a = 13$.

Solução. Neste sistema $n = 43 \times 59 = 2537$. Note que, como 13 é primo, $\text{mdc}(13, 42 \times 58) = 1$. Traduzindo as letras no seu valor numérico, $H \rightarrow 07$, $E \rightarrow 04$, $L \rightarrow 10$ e $P \rightarrow 14$. A mensagem corresponde então ao número $x = 07041014$. Se aplicarmos já a chave pública u a x obtemos uma mensagem só com quatro algarismos: $u(x) = 07041014^{13} \pmod{2537} = 1507$. Para manter o número de algarismos na mensagem encriptada, como $n = 2537$ tem quatro algarismos, agrupam-se os algarismos de x em blocos de quatro e só depois se aplica u a cada um desses blocos¹⁹:

$$0704 \xrightarrow{u} 704^{13} \pmod{2537} = 0981,$$

$$1014 \xrightarrow{u} 1014^{13} \pmod{2537} = 1175.$$

A mensagem encriptada é então 0981 1175.

O receptor quando recebe a mensagem descripta-a com a ajuda da chave privada b que só ele conhece:

$$v(u(x)) = u(x)^b \pmod{n}.$$

Teste. Descodifique a mensagem seguinte, recebida usando o sistema RSA do exemplo anterior: 2128 2431.

Solução. Temos que resolver a congruência $13b \equiv_{42 \times 58} 1$ para determinar a chave privada b :

$$42 \times 58 = 2436 = 13 \times 187 + 5$$

$$13 = 5 \times 2 + 3$$

$$5 = 3 \times 1 + 2$$

$$3 = 2 \times 1 + 1$$

¹⁹Portanto, deveremos ter o cuidado de ter sempre $x < n$.

donde

$$\begin{aligned}
 1 &= 3 - 2 \times 1 \\
 &= 3 - (5 - 3 \times 1) \\
 &= 3 \times 2 - 5 \\
 &= (13 - 5 \times 2) \times 2 - 5 \\
 &= 13 \times 2 - 5 \times 5 \\
 &= 13 \times 2 - (2436 - 13 \times 187) \times 5 \\
 &= 13 \times (2 + 187 \times 5) - 2436 \times 5 \\
 &= 13 \times 937 - 2436 \times 5.
 \end{aligned}$$

Portanto $b = 937$. Então $2128^b \bmod 2537 = 2128^{937} \bmod 2537 = 1718$ e $2431^b \bmod 2537 = 2431^{937} \bmod 2537 = 1314$, pelo que a mensagem original é, na versão numérica, 1718 1314, ou seja STOP.

Já sabemos como encriptar e desencriptar mensagens no sistema RSA. Falta assegurar, como tínhamos anunciado, que o RSA satisfaz a propriedade (P1), isto é, que a desencriptação v é de facto inversa da encriptação u :

Proposição. *Sejam p e q primos distintos, $n = pq$ e $m = (p-1)(q-1)$. Se a e b são inteiros tais que $ab \equiv_m 1$, então $v(u(x)) = x$ para qualquer inteiro $x < p, q$.*

Prova. Uma vez que $v(u(x)) = v(x^a \bmod n) = (x^a \bmod n)^b \bmod n = x^{ab} \bmod n$, e $ab = k(p-1)(q-1) + 1$ para algum inteiro k , temos

$$v(u(x)) = x^{k(p-1)(q-1)+1} \bmod n = ((x^{(p-1)(q-1)})^k \times x) \bmod n.$$

Mas como x é menor do que p e q , não é divisível por p e q logo, pelo Teorema de Fermat-Euler (b), $(x^{(p-1)(q-1)})^k \bmod n = 1$. Portanto $v(u(x)) = x \bmod n = x$. \square

Basta assim usarmos mensagens com número x inferior a p, q para termos a certeza que a função de desencriptação v recupera a mensagem original.

Como é que o processo de troca de mensagens secretas entre a Alice e o Bruno se desenrola na realidade? O Bruno, o receptor, faz o seguinte:

- (1) escolhe dois números primos p e q ,
(*pode ser difícil quando se procuram números p, q muito grandes*)
- (2) calcula os produtos $n = pq$ e $m = (p-1)(q-1)$,
(*muito fácil*)
- (3) escolhe $a \in \mathbb{Z}_m$ (a chave pública) tal que $\text{mdc}(a, m) = 1$,
(*fácil: conhecemos um algoritmo eficaz para calcular o mdc de 2 números*)
- (4) usando o algoritmo de Euclides, determina $b \in \mathbb{Z}_m$ (a chave privada) tal que $ab \equiv_m 1$.
(*fácil: conhecemos um algoritmo eficaz para calcular o inverso de um elemento em \mathbb{Z}_m*)
- (5) Envia os valores de n e a para a Alice, mantendo a chave privada b só do seu conhecimento.
(a partir daqui, não havendo garantias de segurança no canal de comunicação, os valores de n e a passam a ser eventualmente públicos.)

A Alice tem agora os elementos para encriptar as suas mensagens com a função u e enviá-las ao Bruno. Como só este conhece o valor de b , só ele poderá decifrar a mensagem aplicando a função v .

E que trabalho tem que fazer uma terceira pessoa mal intencionada, que conhece só a função de encriptação, para descriptar uma mensagem?

- (1) factorizar o número n para recuperar os primos p e q ,
(*pode ser muito difícil quando n é muito grande*)
- (2) usando o algoritmo de Euclides, determinar $b \in \mathbb{Z}_m$ (a chave privada) tal que $ab \equiv_m 1$,
(*fácil: conhecemos um algoritmo eficaz para calcular o inverso de um elemento em \mathbb{Z}_m*).

Vemos assim que a diferença principal é:

- para criar o código é preciso encontrar dois números primos p, q ;
- para decifrar o código é preciso factorizar o produto $n = pq$.

Comparemos a dificuldade de ambas as operações. Actualmente, os grandes computadores utilizam números primos com mais de 80 algarismos. Claramente, se um número p de 80 algarismos não é primo, podemos escrever $p = a \times b$ e é impossível que ambos os factores tenham mais de 40 algarismos. Portanto, um dos factores tem menos de 40 algarismos. Assim, para saber se p é primo, basta dividir p por todos os números de 40 algarismos ou menos. E se o número não é primo, recomeçar o trabalho com um outro número, e assim sucessivamente, até achar finalmente um número primo.

Existe uma fórmula famosa que dá uma aproximação do número de números primos inferiores a um certo número n :

Existem cerca de $\frac{n}{\ln n}$ números primos inferiores a n .

Isto implica que, entre os números de 80 algarismos, aproximadamente um em cada 185 números é primo. Então, para encontrar dois números primos de 80 algarismos, o Bruno tem que fazer cerca de

$$2 \times 185 \times 10^{40} = 370 \times 10^{40}$$

operações.

Quando uma terceira pessoa mal intencionada conhece $n = pq$, sabendo que p e q são números primos de 80 algarismos, tem que dividir n por todos os números de 80 algarismos para descobrir os factores p, q . Há 10^{80} números de 80 algarismos ou menos, e 10^{79} números de 79 algarismos ou menos. Então há

$$10^{80} - 10^{79} = 10^{79}(10 - 1) = 9 \times 10^{79}$$

números de exactamente 80 algarismos. Em conclusão, o trabalho da pessoa mal intencionada é

$$\frac{9 \times 10^{79}}{370 \times 10^{40}} = \frac{9}{37} \times 10^{38} \approx 2,4 \times 10^{37}$$

vezes mais difícil do trabalho do Bruno. Isto significa que se o computador do Bruno gastar um segundo a encontrar os primos p, q , um computador do mesmo tipo tem que trabalhar durante aproximadamente 10^{37} segundos para quebrar o código. É muito tempo?

A Terra tem cerca de 3.500.000.000 anos, ou seja,

$$3.500.000.000 \times 365 \times 24 \times 60 \times 60 = 110.376.000.000.000.000 \approx 1,1 \times 10^{17}$$

segundos. Então 10^{37} segundos são 10^{20} vezes a idade da Terra!!!

É claro que o exemplo de RSA que apresentamos na página 95 é (matematicamente) inseguro, um intruso facilmente quebraria o sistema: conhecendo $n = 2537$, facilmente obteria a sua factorização prima $pq = 43 \times 49$, calcularia $(p - 1)(q - 1) = 2436$, e usaria o algoritmo de Euclides para descobrir a chave privada $b = 937$.

É aqui que reside a grande segurança do sistema RSA: a aparente dificuldade em resolver este problema, desde que p e q sejam números com muitos algarismos. Neste caso, mesmo sendo pública a informação de n e a , um intruso não deverá ser capaz de descobrir o expoente b de descriptação²⁰.

Exemplo. O Bruno começa por procurar dois primos grandes p e q , por exemplo com 80 algarismos. Com `nextprime(r)` calcula o menor primo maior do que o input inteiro r :

```
> r := rand(10^80);
> s := rand(10^80);
> p := nextprime(r);
> q := nextprime(s);

r := 19669081321110693270343633073697474256143563558458718976746753830538032062222085
s := 74121768604305613921745580037409259811952655310075487163797179490457039169594160
p := 19669081321110693270343633073697474256143563558458718976746753830538032062222257
q := 74121768604305613921745580037409259811952655310075487163797179490457039169594213
```

O uso da função `rand` na selecção dos primos tem a intenção de os tornar difíceis de adivinhar. Em seguida, calcula $n = pq$ e $m = (p - 1)(q - 1)$.

```
> n := p*q;

n := 145790709434263657193410815968586298032651591491182486164339752298049755073623
0615496046802186876835611836753440525199587698019954839165932427842278373706998741
```

Este valor de n com 160 algarismos permite encriptar mensagens até 80 letras num só bloco.

```
> m := (p-1)*(q-1);

m := 145790709434263657193410815968586298032651591491182486164339752298049755073623
0521705196876770569643522623642333791131491479151420633025388494521283302475182272
```

Em seguida, decide a escolha de a . Como será um valor público, não se preocupa em gerar números aleatoriamente. A única preocupação é que satisfaça $\text{mdc}(a, m) = 1$. Por exemplo,

²⁰A não ser que descubra um algoritmo de factorização que torne o problema realizável em tempo útil, o que os matemáticos acreditam (conjecturam) não existir. Mas, até hoje, ninguém foi capaz de provar isso. Por esta razão a factorização, para a qual se julga não existir algoritmo polinomial, é o calcanhar de Aquiles do RSA.

pode fazer $a = 2^{16} + 1 = 65537$. Para verificar que se trata de um expoente de encriptação válido, verifica o $\text{mdc}(a, m)$:

```
> isprime(a);
> gcd(a,m);

      true
      1
```

Agora calcula b tal que $ab \equiv_m 1$:

```
> b := eval(1/a mod m);

b := 3418029892209684747206550784072094342541910223632480735943177585271731215506077
78293183240178522095499109087453784896094825475099226794560236481979918863102913
```

Está pronto para definir as função de encriptação u e desencriptação v :

```
> u := (x,a,n) -> Power(x,a) mod n;
> v := (x,b,n) -> Power(x,b) mod n;
```

Verifiquemos num exemplo (mensagem $x = "5"$) que os procedimentos u e v são inversos um do outro:

```
> y := u(5,a,n); x :=v(y,b,n);

y := 4466998265285704577278497028478824560110639588454365754063648457739348831857859
04969215738362722324430978629195687422700096164664107349103915395164169538874261
x := 5
```

Teste. Descodifique a mensagem 1445271342077333850810587930721246119637276300086542923991011323094820891531712659656668032513734254782932933643187458814460659880338609653396403575967077856790 recebida pelo Bruno.

Solução.

```
> x :=v(1445271342077333850810587930721246119637276300086542923991011323094820
8915317126596566680325137342547829329336431874588144606598803386096533964035759
67077856790,b,n);

x := 417181903041104171816191819160017030817021604180017
```

Como o primeiro par de algarismos, 41, não corresponde a nenhuma letra (cf. tabela da página 57), o par original deverá ser 04 (o *Maple* não escreveu o 0), ou seja, é a letra E. Continuando, $17 \rightarrow S$, $18 \rightarrow T$, $19 \rightarrow U$, etc. A mensagem original é

“ESTUDEMESTRUTURASDISCRETAS”.

Observe que, em geral, no sistema RSA assume-se que quase tudo é do conhecimento público, incluindo a forma da função encriptadora. Isto significa que um intruso que intersecta uma mensagem RSA sabe que esta foi formada com a função $u(x) = x^a \bmod n$, e conhece os valores a e n . A vantagem desta informação ser pública reside no facto da Alice e do Bruno para comunicarem entre si numa linha de comunicação insegura não precisarem de pensar numa maneira de trocarem entre si secretamente o expoente de encriptação a e o módulo n . Só a chave privada b nunca pode circular entre ambos, pelo canal de comunicação, para que não possa ser interceptada.

Leituras suplementares. (1) Até agora utilizámos um método muito simples de conversão de letras em números, que representa A pelo número 0, B pelo número 1, etc., até Z. Este método tem uma desvantagem óbvia: não funciona conjuntamente com maiúsculas e minúsculas, com espaços, acentos e outros caracteres. Existe um método muito utilizado de conversão de caracteres no código ASCII, usado pela maioria dos computadores, e que está implementado no **Maple**, permitindo a conversão de cadeias de caracteres alfanuméricos em inteiros e vice-versa. A instrução é

```
convert("xxx", bytes).
```

```
> u := convert("Bom dia, a vossa missão para hoje é codificar esta mensagem.",
bytes);
u := [66, 111, 109, 32, 100, 105, 97, 44, 32, 97, 32, 118, 111, 115, 115, 97, 32, 109, 105, 115, 115,
227, 111, 32, 112, 97, 114, 97, 32, 104, 111, 106, 101, 32, 233, 32, 99, 111, 100, 105, 102, 105, 99,
97, 114, 32, 101, 115, 116, 97, 32, 109, 101, 110, 115, 97, 103, 101, 109, 46]
```

Em sentido inverso:

```
> convert([66, 111, 109, 32, 100, 105, 97, 44, 32, 97, 32, 118, 111, 115, 115,
97, 32, 109, 105, 115, 115, 227, 111, 32, 112, 97, 114, 97, 32, 104, 111, 106,
101, 32, 233, 32, 99, 111, 100, 105, 102, 105, 99, 97, 114, 32, 101, 115, 116,
97, 32, 109, 101, 110, 115, 97, 103, 101, 109, 46], bytes);
"Bom dia, a vossa missão para hoje é codificar esta mensagem."
```

É possível implementar um procedimento em **Maple** que traduza mensagens alfanuméricas numa sequência de inteiros, usando a conversão para ASCII, concatene essa sequência num só inteiro grande pronto a ser utilizado pelo algoritmo RSA e, em sentido inverso, converta um inteiro na respectiva mensagem escrita. Se estiver interessado, consulte o texto²¹ `conversaoRSA2.mws` de Mike May (2002).

(2) O sistema RSA tem resistido a ataques de criptoanalistas, à custa do aumento da dimensão das chaves, mas é necessário ir acompanhando os desenvolvimentos mais recentes. Apesar da matemática subjacente ser há muito conhecida, a cifra RSA surgiu apenas nos anos 70 porque é aplicável apenas com números primos de grande dimensão e só nos anos 70 apareceram computadores potentes de custo aceitável. Ataques mais conhecidos em 2006:

²¹Na página da disciplina.

- Números até 100 bits consegue-se quebrar, com PCs.
- Em computação paralela são conhecidos ataques até 640 bits (actualmente recomenda-se o uso de números RSA-1024, ou RSA-2048)²².

Qual é a complexidade do algoritmo de factorização do número n em primos pq ?

Factorização à força bruta: testar todos os primos até $n/2$ (é de complexidade $O(\sqrt{n})$).

Exemplo. $n := 408508091$.

1.	Divisível por 3? Não!
2.	Divisível por 5? Não!
3.	etc.
:	:
2099.	Divisível por 18 313 (é o 2099º primo)? Sim, está identificado o primo p .
2100.	$q = n/18313 = 22307$.

Demorou 2099 passos, e n só tem 9 algarismos! Imagine RSA-640 com 193 algarismos decimais...

A empresa norte-americana *RSA Security*²³ submete a concurso a factorização de números RSA com prémios até 200 mil dólares (para o caso RSA-2048):

- Primeiro prémio (100 dólares) atribuído em Abril 1994, pela factorização de números RSA-129.
- O prémio mais elevado (20 000 dólares) foi ganho em Novembro 2005 na factorização do RSA-640

[3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609](#)

por F. Bahr, M. Boehm, J. Franke, T. Kleinjung (Univ. Bona, Alemanha). Os factores são

[1634733645809253848443133883865090859841783670033092312181110852389333100104508151212118167511579](#)

e

[190087128166482211312685157393541397547189678996851549366638539088027103802104498957191261465571](#)

Os cálculos foram efectuados durante 540 dias por um conjunto de 80 AMD64 Opteron (CPU que equipa cerca de 10% dos supercomputadores mais rápidos do mundo).

²²A notação RSA-xxxx refere-se a um número RSA com tamanho xxxx em bits.

²³www.rsa.com/rsalabs/node.asp?id=2093.

Desafio. Se conseguir descobrir a factorização do RSA-704

```
74037563479561712828046796097429573142593188889231289084936232638972765034028266
27689199641962511784399589433050212758537011896809828673317327310893090055250511
6877063299072396380786710086096962537934650563796359
```

terá 20 valores à disciplina e pode candidatar-se ao prémio de 30 000 dólares da *RSA Security*.

Exemplos de chaves públicas usadas em algumas páginas web.

```
> with(StringTools):
```

Chave pública Departamento de Matemática (tamanho: 140 Bytes / 1120 Bits)

```
> nDMUC := "30 81 89 02 81 81 00 db 35 3c 03 49 fc e0 48 e4 b6 7c 55 66 f3 52 08
39 b9 d8 bf eb 9c c7 e8 7a 32 54 fc 88 66 19 de a0 08 b1 19 ad a0 34 75 0c 2b 0d
f5 6d 3b 9d f4 78 2e 2e fe 45 d3 7e b5 ff 7c f6 9a 3b d7 13 46 a9 e1 ab 0b 01 d8
d8 3c 65 d7 ce a3 e7 32 c3 59 54 97 54 b7 a4 6e be 07 61 25 0d 32 04 3a 99 15 19
23 9d 97 61 e1 66 5d b7 ef 81 e9 1e cd bc 25 fd 39 9b 74 6a 86 09 07 9a 98 bd 45
f2 ba 84 a1 02 03 01 00 01":
> nDMUC := SubstituteAll(nDMUC, " ", ""):
```

Conversão da representação hexadecimal de um número para a sua representação decimal

```
> nDMUC := convert(nDMUC, decimal, hexadecimal);
```

```
nDMUC := 26986751662544233897390996989562786998209640195002153770034468141230169
50163931124306976585375250715400911398515483295152423812242464904341510349824376
57607908609525327637173783415854826421482431563077009840890804022964491227968726
58542504958657945923684483016763566353046590571882008173675016517836292426828436
78677672254248775317520385
```

Chave pública VISA eCommerce (tamanho: 270 Bytes / 2160 Bits)

```
> nVISA := "30 82 01 0a 02 82 01 01 00 af 57 de 56 1e 6e a1 da 60 b1 94 27 cb 17
db 07 3f 80 85 4f c8 9c b6 d0 f4 6f 4f cf 99 d8 e1 db c2 48 5c 3a ac 39 33 c7 1f
6a 8b 26 3d 2b 35 f5 48 b1 91 c1 02 4e 04 96 91 7b b0 33 f0 b1 14 4e 11 6f b5 40
af 1b 45 a5 4a ef 7e b6 ac f2 a0 1f 58 3f 12 46 60 3c 8d a1 e0 7d cf 57 3e 33 1e
fb 47 f1 aa 15 97 07 55 66 a5 b5 2d 2e d8 80 59 b2 a7 0d b7 46 ec 21 63 ff 35 ab
a5 02 cf 2a f4 4c fe 7b f5 94 5d 84 4d a8 f2 60 8f db 0e 25 3c 9f 73 71 cf 94 df
4a ea db df 72 38 8c f3 96 bd f1 17 bc d2 ba 3b 45 5a c6 a7 f6 c6 17 8b 01 9d fc
19 a8 2a 83 16 b8 3a 48 fe 4e 3e a0 ab 06 19 e9 53 f3 80 13 07 ed 2d bf 3f 0a 3c
55 20 39 2c 2c 00 69 74 95 4a bc 20 b2 a9 79 e5 18 89 91 a8 dc 1c 4d ef bb 7e 37
0b 5d fe 39 a5 88 52 8c 00 6c ec 18 7c 41 bd f6 8b 75 77 ba 60 9d 84 e7 fe 2d 02
03 01 00 01":
```