Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Matemática

SMA 180 - Matemática Discreta I

Eugenio Massa

13 de novembro de 2023

Estas notas são escritas para ajudar no curso do ICMC-USP, SMA180 - Matemática Discreta I.

As notas pretendem cobrir quase inteiramente a ementa do curso, mesmo que nem sempre entrarão em todos os detalhes (não dispensando então a frequência nas aulas nem o estudo individual).

No final são citadas as principais referências usadas. Note-se porém que muitas vezes a notação ou até alguns detalhes, foram trocados com respeito às referências para obter uma melhor clareza (pelo menos na minha opinião) e para ter uniformidade de notação ao longo das notas.

Agradeço se me enviarem comentários ou correções, no endereço eugenio@icmc.usp.br

Sumário

Capítulo 1

Lógica

Neste capítulo veremos uma introdução de lógica, que será útil para desenvolver os assuntos seguintes.

1.1 Conectivos lógicos

Os **conectivos lógicos** servem para juntar afirmações. Vejamos em particular os seguintes 4:

Nome:	AND	OR	XOR	NOT
em port:	e	ou	ou exclusivo	não
Símbolo	$\wedge$	$\vee$	$\oplus$	$-$ " " (ou $\overline{\text{" "}}$)[1]
em C	&&	||		!
	binário			unário

Consideremos algumas afirmações A1,A2,..; podemos construir novas afirmações juntando elas com os conectivos:

$$A3 := A1 \wedge A2$$

é uma nova afirmação que é verdadeira se e só se A1 e A2 são verdadeiras,

$$A4 := \overline{A1}$$

é uma nova afirmação que é verdadeira se e só se A1 é falsa.

Os conectivos AND, OR, XOR[2] são **binários**, isto é, juntam duas afirmações, enquanto o NOT é **unário**: apenas age sobre uma afirmação.

[1] No livro [SDK15a] usa "$-A$" para negar A, aqui usaremos prevalentemente "$\overline{A}$".

[2] Cuidado: OR e XOR na língua falada correspondem ambos a "ou"; a diferença em geral está no contexto: por exemplo
"vai ou fica?" é XOR,
"para ganhar precisa habilidade ou sorte" é OR.

A nova afirmação construída será verdadeira (V) ou falsa (F) dependendo dos valores V/F das afirmações que a compõem e da definição do conectivo.

Abaixo estão as definições dos conectivos acima, através de suas **tabelas de verdade**, as quais definem, em cada quadrado, o valor da afirmação composta em função dos valores de cada afirmação.

Af1 AND Af2:

Af1 ∧ Af2		
Af1 \ Af2	V	F
V	**V**	**F**
F	**F**	**F**

(1.1)

Af1 OR Af2:

Af1 ∨ Af2		
Af1 \ Af2	V	F
V	**V**	**V**
F	**V**	**F**

(1.2)

Af1 XOR Af2:

Af1 ⊕ Af2		
Af1 \ Af2	V	F
V	**F**	**V**
F	**V**	**F**

(1.3)

NOT Af1 :

$\overline{\text{Af1}}$	
Af1	$\overline{\text{Af1}}$
V	**F**
F	**V**

(1.4)

Definição 1.1. *Dadas duas afirmações complexas, elas serão* **equivalentes** *se elas forem V ou F nas mesma situações: uma forma de verificar isso é comparando a tabela de verdade delas.*

Exemplo 1.2. $\overline{a} \vee (b \wedge c)$ é equivalente a $\overline{a \vee (b \wedge c)}$?
Vamos pôr nas linhas todas as possíveis combinações de V/F para a,b,c e ver o valor V/F

das afirmações compostas:

a	b	c	$b \wedge c$	$\overline{a}$	$\overline{a} \vee (b \wedge c)$	$a \vee (b \wedge c)$	$\overline{a \vee (b \wedge c)}$
V	V	V	V	F	V	V	F
V	V	F	F	F	F	V	F
V	F	V	F	F	F	V	F
V	F	F	F	F	F	V	F
F	V	V	V	V	V	V	F
F	V	F	F	V	V	F	V
F	F	V	F	V	V	F	V
F	F	F	F	V	V	F	V

Comparando as colunas última e antepenúltima vemos que as afirmações não são equivalentes (por exemplo na primeira linha). ◁

Observação 1.3. Podemos obter uma afirmação que reproduza uma qualquer tabela de verdade simplesmente juntando com OR cada combinação correspondente aos V da tabela, escrita usando AND e NOT. Por exemplo, se numa tabela correspondente a uma afirmação composta por 3 afirmações A,B,C, o V está apenas quando A,B são V e C é F e quando A é V e B,C são F, então uma forma de escrever a afirmação é $(A \wedge B \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge \overline{C})$. ◁

1.1.1 Propriedades dos conectivos AND OR XOR NOT

Uma outra forma de comparar afirmações compostas é usando propriedades dos conectivos (abaixo "="significa "é equivalente a").

Leis distributivas AND/OR:[3]

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \qquad (\wedge \text{ distribui sobre } \vee), \tag{1.5}$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \qquad (\vee \text{ distribui sobre } \wedge). \tag{1.6}$$

Leis de De Morgan

$$\overline{(a \vee b)} = \overline{a} \wedge \overline{b} \tag{1.7}$$
$$\overline{(a \wedge b)} = \overline{a} \vee \overline{b} \tag{1.8}$$

[3] As leis distributivas se parecem às que temos com soma e produto, mas aqui elas valem nas duas formas acima, diferente do caso soma produto onde não é verdadeira $a + (b \cdot c) = (a + b) \cdot (a + c)$ mas apenas $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$, isto é ($\cdot$ distribui sobre $+$, mas $+$ não distribui sobre $\cdot$).
Cuidado: $\vee$ não distribui sobre $\oplus$: verifique.
$\wedge$ distribui sobre $\oplus$?

Dupla negação:

$$\overline{\overline{a}} = a. \tag{1.9}$$

Expressão para XOR:

$$a \oplus b = (a \vee b) \wedge \overline{a \wedge b} = (a \wedge \overline{b}) \vee (\overline{a} \wedge b). \tag{1.10}$$

Exercício 1.4. Prove as propriedades usando a tabela de verdade. ★

Observação 1.5. As leis de de Morgan são análogas às homônimas leis para conjuntos, substituindo AND por $\cap$, OR por $\cup$ e NOT por complementar. De fato, são equivalentes: é suficiente definir a afirmação Af_i="pertence ao conjunto A_i"e verificar que
– $Af_1 \wedge Af2$="pertence ao conjunto $A_1 \cap A_2$",
– $Af_1 \vee Af2$="pertence ao conjunto $A_1 \cup A_2$",
– $NOTAf_1$="pertence ao conjunto A_1^c".
Também podem ser provadas pela tabela de verdade.

As leis de de Morgan são importantes pois mostram como negar uma afirmação com AND ou com OR: negar $A \wedge B$ é o mesmo que negar as duas afirmações e pôr OR no meio, isto é, é suficiente que seja F uma das duas para que seja F a afirmação "ambas são V".
Inclusive isso pode ser visto pelas tabelas de verdade (1.1-1.2): negar as afirmações significa trocar as linhas entre si, e as colunas entre si, negar tudo significa trocar V/F entre eles. ◁

1.2 Implicações

Vejamos mais dois importantíssimos conectivos lógicos binários:

Af1 implica Af2:

Af1 $\Longrightarrow$ Af2		
Af2 \ Af1	V	F
V	**V**	**V**
F	**F**	**V**

(1.11)

Af1 se e só se Af2:

Af1 $\Longleftrightarrow$ Af2		
Af2 \ Af1	V	F
V	**V**	**F**
F	**F**	**V**

(1.12)

- $A \Longrightarrow B$ pode ser lida como "A implica B", "se A então B", "A, só se B".
  Para ficar claro, pense na tabela de verdade: sempre que tem A deve ter B: "se A então B"; quando B é F, A deve ser F, logo "A, só se B".

- A afirmação $A \iff B$ significa que $A \implies B$ e $B \implies A$ (verifique isso).

  $A \iff B$ pode ser lida como "A e B são equivalentes", "A se e só se B".
  A segunda formulação vem de juntar as duas formas vistas acima de ler $A \implies B$ e $B \implies A$.

O fato da segunda coluna de $\implies$ ser toda V pode ser entendido se a gente pensar a afirmação $A \implies B$ como "toda vez que A, também B"[4], assim é claro que quando A é F, independente de como é B, a afirmação será verdadeira, ou dito de outra forma, $A \implies B$ só será F se A acontece e B não.

Observação 1.6. Observe que juntando os conectivos vistos temos praticamente esgotado todas as possíveis tabelas de verdade para conectivos binários: deixando os triviais em que o resultado independe de uma das afirmações, as tabelas com 2V e 2F são as de XOR e $\iff$ (aliás, $\oplus = \overline{\iff}$, no sentido que $A \oplus B$ equivale a $\overline{A \iff B}$), enquanto as tabelas com 3V são as de $\implies$ e de OR (observe que podemos transpor a tabela trocando a ordem das afirmações) e as com 3 F são a de AND e $\overline{\implies}$, no sentido de $\overline{A \implies B}$). $\lhd$

1.2.1 Propriedades das implicações

Valem as seguintes propriedades

$$(a \implies b) = \overline{a} \vee b \tag{1.13}$$
$$(a \implies b) = (\overline{b} \implies \overline{a}) \tag{1.14}$$
$$(a \iff b) = (\overline{b} \iff \overline{a}) \tag{1.15}$$

A propriedade (1.13) é bem útil (por exemplo para poder aplicar as propriedades dos conectores AND, OR, NOT): verifique ela.

A propriedade (1.14), introduz a chamada **afirm. contrapositiva** (ou contranominal): chamamos contrapositiva de $(a \implies b)$ a afirmação $(\overline{b} \implies \overline{a})$: a propriedade diz que elas são equivalentes!! (verifique). Uma forma de provar isso, usando (1.13) e (1.9), é a seguinte: $(a \implies b) = \overline{a} \vee b = \overline{a} \vee \overline{\overline{b}} = (\overline{b} \implies \overline{a})$.

CUIDADO: pelo contrário, $(a \implies b)$ não é equivalente (verifique) a $(b \implies a)$, que é dita **afirm. reciproca**).

Observação 1.7. Podemos usar "$\iff$" no lugar do "=" entre afirmações, para dizer que são equivalentes; ex: $(\overline{a \wedge b}) \iff (\overline{a} \vee \overline{b})$. $\lhd$

[4] É importante destacar que o significado de "implica" deve sere entendido estritamente como o conectivo cuja tabela é a definida acima, mesmo que em alguns casos o significado não seja exatamente o da língua comum. Por exemplo "1+1=3 implica que hoje chove" é uma afirmação V, já que "1+1=3" é F, mesmo que não tenha nenhuma relação de causa-efeito como frequentemente é subentendido na palavra implicar.

1.3 Quantificadores: ($\forall$, $\exists$)

Os quantificadores são instrumentos importantes para construir afirmações.
Consideraremos afirmações que dependem de uma variável, por exemplo:

Af1(x)="$x > 0$", Af2(n)="n é par".

Neste caso então a afirmação não tem valor V/F a priori, mas depende do valor da variável:

Af1(3) é V, Af2(3) é F.

Podemos porém construir novas afirmações usando os seguintes quantificadores:[5]

- **quantificador universal:** $\forall$ (para todo).

  A afirmação A="$\forall\, x$ vale Af(x)" é :
  VERDADEIRA se Af(x) vale para todo x
  FALSA se Af(x) é falsa para pelo menos um x (existe um x tal que Af(x) é falsa)

- **quantificador existencial:** $\exists$ (existe).

  A afirmação B="$\exists\, x$ tal que vale Af(x)" é :
  VERDADEIRA se Af(x) vale para pelo menos um x
  FALSA se Af(x) é falsa para todo x

Exemplo 1.8.
A afirmação "$\forall\ n \in \mathbb{N},\ n$ é par" é F,
A afirmação "$\exists\ n \in \mathbb{N} :\ n$ é par" é V,

A afirmação "$\forall\ n \in \mathbb{N},\ 2n$ é par" é V,
A afirmação "$\exists\ n \in \mathbb{N} :\ 2n$ é ímpar" é F.

Exemplo 1.9. Muitas afirmações em matemática são feitas mesmo compondo afirmações e quantificadores. Por exemplo $\lim_{x\to\infty} f(x) = L$ é definida como
"$\forall \varepsilon > 0\ \exists H > 0$ tal que $x \in D_f \wedge x > H \Longrightarrow |f(x) - L| < \varepsilon$".

[5] O livro [SDK15a] escreve:
– $\forall\ n \in \mathbb{N}$ (n é par),
– $\exists\ n \in \mathbb{N}$ (n é par).
Ao ler a afirmação, podemos dizer
"para todo $n \in \mathbb{N}$ vale (que) n é par"
"existe $n \in \mathbb{N}$ tal que n é par"

1.3.1 Negação de afirmações com quantificadores

É muito importante saber negar uma afirmação com quantificadores: a regra é que precisa trocar o quantificador e negar a afirmação com a variável, isto é :

$$\overline{\forall\, x \in X \text{ vale } p(x)} \iff \exists\, x \in X :\ \overline{p(x)}\,,$$

$$\overline{\exists\, x \in X :\ p(x)} \iff \forall\, x \in X \text{ vale } \overline{p(x)}\,.$$

Para negar afirmações com mais quantificadores, precisa aplicar a regra um pedaço de cada vez:

Exemplo 1.10. Vamos negar "$\forall\, x\ \exists\, y : p(x,y)$": veja que podemos deixar mais claro usando parenteses: "$\forall\, x\ vale\ (\exists\, y :\ p(x,y))$".
Então negando primeiro o $\forall$ temos:

$$\overline{\forall\, x\ vale\ (\exists\, y :\ p(x,y))} \iff \exists\, x :\ \overline{(\exists\, y :\ p(x,y)}\,.$$

Negando agora a afirmação mais interna

$$\overline{\forall\, x\ vale\ (\exists\, y :\ p(x,y))} \iff \exists\, x :\ (\forall y\ vale\ \overline{p(x,y)}).$$

Analogamente

$$\overline{\exists\, x :\ \forall\, y \text{ vale } p(x,y)} \iff \left(\forall\, x\ \exists\, y :\ \overline{p(x,y)}\right)\,.$$

★

1.3.2 Quantificadores escondidos

Às vezes se escrevem afirmações que contêm quantificadores subentendidos: é importante explicitá-los, por exemplo para poder usar as regras acima na hora de negar afirmações:

Exemplo 1.11. A afirmação "$x > 0 \implies x > 4$" pode ser lida como
"$\forall x > 0\ vale\ x > 4$",
sua negação será então "$\exists x > 0\ tal\ que\ x \leq 4$".
Repare que neste exemplo a primeira afirmação é falsa e a sua negação é verdadeira.
★

Exercício 1.12. Negue a afirmação "$\lim_{x\to\infty} f(x) = L$".
Solução.
Escrevamos $\lim_{x\to\infty} f(x) = L$ como
"$\forall \varepsilon > 0,\ \exists H > 0 :\ Af$"
onde Af é a implicação "$x \in D_f \wedge x > H \implies |f(x) - L| < \varepsilon$", que podemos enxergar como "$\forall x \in D_f\ vale\ (x > H \implies |f(x) - L| < \varepsilon)$" (quantificador subentendido).
Negando então obtemos
$\exists \varepsilon > 0 :\ \forall H > 0\ vale\ \overline{Af}$

e, lembrando (1.13) e (1.7),
$\overline{Af} = \exists x \in D_f : (|f(x) - L| \geq \varepsilon) \wedge (x > H)$.
Conclusão: "$\lim_{x\to\infty} f(x) = L$" é F quando
"$\exists \varepsilon > 0 : \ \forall H > 0, \ \exists x \in D_f : (|f(x) - L| \geq \varepsilon) \wedge (x > H)$". ★

1.4 Regras de inferência

Vejamos (rapidamente) as regras que são usadas ao fazer uma demonstração.

São chamadas **provas diretas** as demonstrações que usam apenas as seguintes 10 "regras de inferência direta":

Regras para provas diretas

1) Dado um exemplo de $x \in U$: $p(x)$ é falso, concluímos $\overline{\forall x \in U \text{ **vale** } p(x)}$.

2) Dado um exemplo de $x \in U$: $p(x)$ é verdadeiro, concluímos $\exists x \in U : \ p(x)$.

3) se a e b, concluímos $a \wedge b$.

4) se a ou b, concluímos $a \vee b$.

5) se $\overline{a}$ ou b, concluímos $a \Longrightarrow b$.

6) se $a \Longrightarrow b$ e $b \Longrightarrow a$, concluímos $a \Longleftrightarrow b$.

7) se a e $a \Longrightarrow b$, concluímos b. *(inferência direta)*

8) se $a \Longrightarrow b$ e $b \Longrightarrow c$, concluímos $a \Longrightarrow c$. *(transitividade)*

9) se o fato que vale a permite mostrar que vale b, concluímos $a \Longrightarrow b$. *(princípio da prova condicional)*

10) se dado x, o fato que $x \in U$ permite mostrar que vale q(x), concluímos $\forall\, x \in U$ **vale** $q(x)$. *(princípio da generalização universal)*

Observações: Na verdade estas regras são simplesmente as definições e propriedades vistas nas seções anteriores:
– 1,2 são as definições e regras de negação dos quantificadores,
– 3,4,5 são as definições de AND OR $\Longrightarrow$,
– 6 é a definição de $\Longleftrightarrow$,
– 7 pode ser deduzida da tabela de verdade de $\Longrightarrow$,
– 8 é dita **transitividade da implicação**; pode ser verificada analisando a tabela de

verdade de $\Longrightarrow$, ou usando (1.13)[6].
– 9,10 completam as definições de $\Longrightarrow$ e de $\forall$.

São chamadas **provas indiretas** as demonstrações que usam também algumas das regras abaixo:

Regras para provas indiretas:

- **provas contrapositivas:**

  11) se $\overline{b} \Longrightarrow \overline{a}$, concluímos $a \Longrightarrow b$.

- **provas por contradição:**

  12) se supondo a e $\overline{b}$ podemos obter c e $\overline{c}$, concluímos $a \Longrightarrow b$. *(redução ao absurdo)*
  (mais em geral, se supondo a podemos obter c e $\overline{c}$, concluímos $\overline{a}$).

Observações: Estas provas são ditas indiretas porque mostramos uma afirmação que parece não ser a que queremos, mas que de fato é equivalente.
– 11 é a já vista contrapositiva de uma implicação, que já vimos ser equivalente à implicação,
– 12 é consequência de (1.13): se $a \wedge \overline{b}$ implica $c \wedge \overline{c}$ (que é falsa), então também $a \wedge \overline{b}$ é F, isto é $\overline{a} \vee b$ é V.

Exemplo 1.13. Mostremos o seguinte teorema de 3 formas diferentes:
se $a \in \mathbb{Z}$ é ímpar então a^2 é ímpar.
– Prova direta.
Se $a = 2i + 1$ com $i \in \mathbb{Z}$ então $a^2 = 4i^2 + 4i + 1$, logo é ímpar[7].
– Prova por contrapositiva.
Provemos a contrapositiva "se a^2 é par com $a \in \mathbb{Z}$ então a é par";
de fato $a^2 = 2j$ com $j \in \mathbb{Z}$ significa que 2 divide a, logo $a = 2k$ com $k \in \mathbb{Z}$, logo é par.
– Prova por contradição.
Seja c="0 é par" que sabemos ser V;
assumo que $a \in \mathbb{Z}$ é ímpar e a^2 é par,
mas então $a = 2i + 1$ com $i \in \mathbb{Z}$ e $a^2 = 2j$ com $j \in \mathbb{Z}$;
porém $a^2 = 4i^2 + 4i + 1$,
subtraindo concluímos que $0 = 2(j - 2i^2 - 2i) + 1$;
como o que está em parenteses é um inteiro concluímos $\overline{c}$.
Chegamos na contradição $c \wedge \overline{c}$ logo provamos o teorema.

[6] De fato, $(a \Longrightarrow b) \wedge (b \Longrightarrow c)$ é equivalente a $(\overline{a} \vee b) \wedge (\overline{b} \vee c)$. Observando esta afirmação vemos que se ela é verdadeira, então
– se b é F necessariamente $\overline{a}$ é V,
– se $\overline{b}$ é F necessariamente c é V,
concluímos que $\overline{a} \vee c$, que equivale a $a \Longrightarrow c$.

[7] Um ímpar se escreve como o dobro de um inteiro mais 1, um par como dobro de um inteiro.

Observação 1.14. Os exemplos acima são muito simples e as três formas de demonstrar se parecem muito, mas em casos mais complicados uma forma pode ser bem mais fácil que outras.

1.5 Indução

Outra técnica de demonstração muito útil é a indução.

A demonstração por indução é baseada no princípio abstrato abaixo:

Princípio de Indução (fraca)
Seja $U \subseteq \mathbb{N}$ [8]um conjunto com as propriedades:

- $1 \in U$,
- $\forall\, k \in \mathbb{N}$ vale "se $k \in U$ então $k+1 \in U$".

Então $U = \mathbb{N}$.

O Princípio acima é bastante intuitivo, de fato não é propriamente um teorema, mas faz parte dos axiomas que definem os números naturais.

Na prática usaremos o PdI para provar afirmações que dependem de naturais (ou às vezes de inteiros), por isso é conveniente reformular o princípio na seguinte forma:

Princípio de Indução (fraca) (para afirmações)
Seja $p(n)$ uma afirmação sobre $n \in \mathbb{N}$ tal que:

- $p(1)$ é verdade,
- $\forall\, k \in \mathbb{N}$ vale "se $p(k)$ é verdade então $p(k+1)$ é verdade".

Então $p(n)$ é verdade $\forall\, n \in \mathbb{N}$.

Etapas de uma prova por indução (fraca):
Vejamos como fazer uma demonstração por indução: nela sempre tem dois passos fundamentais:

- o **caso base**, isto é, provar que p(1) é V;
- o **passo de indução**, isto é, provar a implicação $p(k) \Longrightarrow p(k+1)$. Esta segunda passagem é em geral a mais delicada pois o que precisa mostrar não é que $p(k)$ e/ou $p(k+1)$ sejam V/ F, mas apenas que a implicação é verdadeira. Precisará então

[8]Nestas notas sempre consideraremos o conjunto $\mathbb{N}$ dos números naturais sendo composto por $1, 2, 3, \ldots$. Caso nos interesse incluir 0 escreveremos $\mathbb{N} \cup \{0\}$.

  - assumir a **Hipótese de indução**[9] $p(k)$ para um genérico $k \in \mathbb{N}$;
  - provar $p(k+1)$ usando apenas $p(k)$ (e regras de inferência).

- Provados os dois passos acima, podemos então concluir pelo princípio de indução, que $p(n)$ é verdade $\forall\, n \in \mathbb{N}$.

Exemplo 1.15 (Indução fraca)**.** Mostremos por indução a fórmula

$$p(N) = \left(\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \right)$$

- Caso Base) p(1) é V: de fato $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Passo de indução) precisa mostrar que se vale $p(k)$ então deve valer $p(k+1)$.

  Assumimos então como hipótese de indução que para certo $k \in \mathbb{N}$, seja verdade $p(k)$, isto é, $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$.

  Consideremos agora $p(k+1)$, que diz respeito a $\sum_{i=1}^{k+1} i$: posso escrever isso como $\left(\sum_{i=1}^{k} i\right) + k + 1$. A motivação é que dessa forma aparece explicitamente o termo $\sum_{i=1}^{k} i$ que por hipótese de indução vale $\frac{k(k+1)}{2}$.
  Logo $\sum_{i=1}^{k+1} i = \frac{k(k+1)}{2} + k + 1 = (rearrumando) = \frac{(k+1)(k+2)}{2}$ o que é exatamente a expressão da afirmação $p(k+1)$ (substitua $k+1$ a N em $p(N)$).

Tendo provado os dois pontos deduzimos que $p(N)$ *vale* $\forall N \in \mathbb{N}$. ★

1.5.1 Outras formulações

Em algumas situações é mais interessante usar a formulação abaixo, chamada princípio de indução forte:

Princípio de Indução (forte) (para afirmações)
Seja $p(n)$ uma afirmação sobre $n \in \mathbb{N}$ tal que

- $p(1)$ é verdade,
- $\forall\, k \in \mathbb{N}$ vale "se $p(1)...p(k)$ são verdade então $p(k+1)$ é verdade".

Então $p(n)$ é verdade $\forall\, n \in \mathbb{N}$.

[9] Chamamos hipótese de indução pois é algo que estamos assumindo (sem saber a priori se é verdadeiro ou falso) para o fim de provar a implicação do passo de indução.
Em geral será útil na prova de $p(k+1)$ destacar em que momento usamos a hipótese de indução $p(k)$.
Sugestão: ao escrever a hip. de indução, não use a mesma letra que é usada na afirmação que quer provar, ou isso vai gerar confusão: o objetivo é provar por indução $p(n)$ $\forall\, n \in \mathbb{N}$, para isso, como hip. de indução, assuma $p(k)$ para certo $k \in \mathbb{N}$.

A diferença com respeito ao princípio fraco é no passo de indução, onde agora assumimos como hipótese de indução uma hipótese mais forte (não apenas $p(k)$ é V mais todas as $p(i),\ i = 1, ..k$ são V). Por isso às vezes torna-se mais fácil provar o passo de indução nesta forma.

Outras variantes são usadas quando precisamos provar uma afirmação $p(n)$ só a partir de certo n_0:

Variantes (outro ponto inicial)
Seja $p(n)$ uma afirmação sobre $n \in \mathbb{N}$ tal que:

- $p(n_0)$ é verdade,
- $\forall\, k \geq n_0$ vale "se $p(k)$ é verdade então $p(k+1)$ é verdade".
  (ou "se $p(n_0)...p(k)$ são verdade então $p(k+1)$ é verdade".).

Então $p(n)$ é verdade $\forall\, n \geq n_0$.

Exemplo 1.16 (Indução forte)**.** Provemos que todo natural $N \geq 2$ pode ser escrito como produto de primos.

- Caso Base) $p(2)$ é V pois $2 = 2$ e 2 é primo.
- Passo de indução) Assumimos como hipótese de indução que para certo $k \in \mathbb{N}$, seja verdade que todo número de 2 a k pode ser escrito como produto de primos.

  Então temos que

  - se $k+1$ é primo então ele pode ser escrito como produto de primos (ele mesmo)
  - se $k+1$ não é primo então $k+1 = ab$ com $2 \leq a, b \leq k$, mas então tanto a quanto b podem ser escritos (pela hipótese de indução) como produto de primos, e logo $k+1$ também.
- Aplicando indução forte (com início em 2 em vez que em 1) obtemos a propriedade para todo $N \geq 2$.

Repare que saber que k pode ser escrito como produto de primos não ajuda a dizer o mesmo para $k+1$, por isso a indução fraca não teria ajudado neste problema. ★

1.5.2 Provas (facultativo)

As diferentes formulações do princípio de indução mencionadas acima são todas equivalentes (cada uma pode ser provada como consequência de uma das outras).

Elas também são equivalentes ao seguinte

Princípio de boa ordem
Seja $A \subseteq \mathbb{N}$ com $A \neq \emptyset$.
Então A possui um elemento mínimo a_0 (isto é, um elemento a_0 tal que $\forall\, a \in A, a_0 \leq a$).

Exercício 1.17. Mostre a equivalência entre os vários princípio de indução e o da boa ordem. Sugestões.

- A formulação por conjuntos ou por afirmações são equivalentes, apenas considere $U = \{n \in \mathbb{N} :\ p(n) \text{ é } V\}$.
- É trivial que forte implica fraco (porque?).
- Para mostrar que fraco implica forte, assuma que o PdI fraco seja verdadeiro, considere a afirmação $q(N)$="$p(i)$ vale para $i = 1, .., N$", mostre que se $p(1)...p(k) \implies p(k+1)$ então $q(k) \implies q(k+1)$. Então nas hipóteses do PdI forte, $q(N)$ vale para todo $N \in \mathbb{N}$ pelo PdI fraco, logo $p(N)$ também.
- Para mostrar a formulação com início em n_0 é suficiente considerar $q(n) = p(n_0 + n - 1)$ e aplicar o princípio com início em 1.
- mostre que o P. da Boa Ordem implica o PdI fraco por contradição: se nas hipóteses do PdI fraco $U \neq \mathbb{N}$, que acontece com o elemento mínimo de U^c (complementar de U em $\mathbb{N}$)?
- Mostre que o PdI forte implica o P. da Boa Ordem por contradição: se $A \subseteq \mathbb{N}$ com $A \neq \emptyset$ não possui elemento mínimo, mostre que $A^c = \mathbb{N}$ por indução forte, contradizendo que $A \neq \emptyset$.

1.5.3 Indução e recursão

Indução está estritamente relacionada com programação de funções recursivas.

Consideremos o programa abaixo, que calcula o fatorial de um natural, de forma recursiva

```
#include <stdio.h>

int recfac(int n)
{
    if(n==1) {printf("1="); return(1);}
    else  {printf("%d*",n);return(n*recfac(n-1));}
}


int main()
{ int n=12;
```

```
12      printf("%d!=%d",n,recfac(n));
13
14      return 0;
15  }
```

OUTPUT:

```
12*11*10*9*8*7*6*5*4*3*2*1=12!=479001600
```

Veja a estrutura da função recfac:

- quando $n = 1$ apenas retorna 1,
- caso contrário chama a si mesma com o valor $n - 1$ (e multiplica o resultado por n).

Para melhor enxergar a relação entre recursão e indução, imaginemos de querer mostrar, por indução, a afirmação
p(n)="ao chamar recfac(n) a função é chamada um total de n vezes".

- Caso base) $p(1)$ é V já que se $n = 1$ caio no "if" logo retorno sem chamar mais a função (que então foi chamada exatamente uma vez).
- Passo de indução) assumo (Hp de indução) que, para certo $k \in \mathbb{N}$ seja verdade p(k). Que acontece ao chamar p(k+1)? certamente caímos no else (pois $k + 1 > 1$), logo chamamos recfac uma outra vez com parâmetro k, o que faz ela ser chamada k vezes por Hp de indução, chegando a um total de $k + 1$ vezes.
- Por indução então p(n) é V para todo $n \in \mathbb{N}$.

Na demostração acima fica clara a semelhança de estrutura entre função recursiva e indução: o caso base corresponde à linha 5 do programa, enquanto o passo de indução corresponde à linha 6.

1.5.4 Indução estrutural

A **indução estrutural** é usada para mostrar afirmações sobre objetos definidos recursivamente.

A ideia da indução estrutural é reformular a afirmação que queremos mostrar, de forma que dependa de un $n \in \mathbb{N}$, e depois prová-la por indução.

Exemplo 1.18. Seja A um conjunto de listas formadas por zeros e uns definidas assim:

R1) "010" $\in A$ e "1" $\in A$,

R2) se $x \in A$ então "00x", "11x" $\in A$.

Mostremos que

"todo elemento de A possui zeros em número par e uns em número ímpar".

Reformulamos então o problema assim:

$p(n)$="todo elemento de A gerado aplicando n vezes a regra R2 possui zeros em número par e uns em número ímpar" vale para todo $n = 0, 1, 2, ...$

Usemos então indução (começando de $n_0 = 0$).

- Caso Base) $p(0)$ vale pois os únicos elementos gerados aplicando 0 vezes R2 são "010"e "1".

- Passo de indução) Suponha $p(k)$:
  seja x gerado aplicando R2 $k + 1$ vezes, logo $x =$ "$00y$" ou $x =$ "$11y$" onde y foi gerado aplicando R2 k vezes;
  como estamos supondo $p(k)$, y possui zeros em número par e uns em número ímpar, logo x também.

Exercício 1.19. Seja A um conjunto de números definido assim:

R1) $7 \in A$,

R2) se $x, y \in A$ então $x + y \in A$.

Mostre (usando indução estrutural e indução forte!!) que "todo elemento de A é múltiplo de 7".

★

Capítulo 2

Contagem - Análise combinatória

2.1 Introdução

O objetivo deste capítulo é aprender técnicas para **contar os elementos** de certos conjuntos (sempre finitos).

Abaixo alguns exemplos de problemas que aprenderemos a resolver:

Exemplos:

1. Dados dois conjuntos finitos A, B, quantos elementos tem $A \cap B$? e $A \times B$?
2. Dados N pontos não alinhados no plano, quantos segmentos podemos desenhar usando eles como vértices? Quantos triângulos?
3. Dados dois conjuntos finitos A, B, quantas funções $f : A \to B$ existem ? Quantas injetoras? Quantas bijetoras?
4. Quantos subconjuntos S com k elementos existem, num conjunto A de n $(n \geq k)$ elementos?
5. Quantos são os anagramas da palavra GATO? E da palavra TORORO?
6. De quantas maneiras podemos dispor k livros idênticos em n estantes? e se os livros são diferentes?
7. Quantas vezes um programa passa por uma certa instrução?

2.2 Conjuntos e Multiconjuntos

2.2.1 Lembrete sobre conjuntos

Lembramos aqui algumas noções sobre conjuntos que servirão ao longo do capítulo.

Um **conjunto** é uma coleção de objetos (de qualquer tipo: números, letras, frutas, ..., até mesmo outros conjuntos).

Podemos descrever um conjunto de várias formas[1]:
listando os elementos: $A = \{1, 2, 3, 4\}$, $B = \{1, A, \divideontimes\}$,
fornecendo uma regra: $A = \{n \in \mathbb{N} : \ 1 \leq n \leq 4\}$, $C = \{n \in \mathbb{N} : \ n \ par\}$....

Definições:

- A reunião de dois conjuntos é um novo conjunto definido assim:
$$A \cup B = \{x : \ (x \in A) \vee (x \in B)\}.$$
- A interseção de dois conjuntos é um novo conjunto definido assim:
$$A \cap B = \{x : \ (x \in A) \wedge (x \in B)\}.$$
- O complementar de um conjunto é um novo conjunto definido assim:
$$A^c = \{x \in U : \ x \notin A\}.$$
Nesta definição U representa um universo convencional no qual estamos trabalhando, que precisará ser especificado.[2]
- **Cardinalidade de um conjunto C (finito)**: é o número de elementos do conjunto, indicado por $|C|$ ou por $\#C$.
- Dizemos que dois conjuntos A, B são **disjuntos** se não tem elementos em comum: $A \cap B = \emptyset$ ($\emptyset$ indica o conjunto vazio).
- Dizemos que os conjuntos de uma família[3] $\{A_i\}_{i \in I}$ são **mutuamente disjuntos** se por cada dupla de elementos da família eles são disjuntos: $\forall\, i, j \in I : \ i \neq j$ vale $A_i \cap A_j = \emptyset$.
- Para indicar a reunião de todos os conjuntos de uma família $\{A_i\}_{i \in I}$ escrevemos
$$\bigcup_{i \in I} A_i = \{x : \ \exists\, i \in I : \ x \in A_i\}.$$
Para indicar a interseção de todos os conjuntos de uma família $\{A_i\}_{i \in I}$ escrevemos
$$\bigcap_{i \in I} A_i = \{x : \ \forall\, i \in I, \ x \in A_i\}.$$
Usaremos também a notação $\biguplus$[4] para indicar uma **reunião disjunta**: em que os

[1] A convenção é usar chaves: { }.

[2] Por exemplo, se $A = \{1, 2\}$ e tomando por universo os naturais, o complementar de A será $A^c = \{3, 4, ...\}$, tomando por universo a reta real, teríamos $A^c = (-\infty, 1) \cup (1, 2) \cup (2, \infty)$.

[3] "Família de conjuntos" significa de fato um conjunto cujos elementos são conjuntos, usa-se a palavra família para não gerar confusão.
A notação $\{A_i\}_{i \in I}$ indica uma família de conjuntos chamados A_i onde i é um índice que varia no conjunto I. Em geral I será um conjunto finito mas poderia ser qualquer coisa: $\mathbb{N}$, $[a, b]$, $\mathbb{R}$, ...
Por exemplo, $\{A_1, A_2, A_3\} = \{A_i\}_{i \in \{1,2,3\}}$.

[4] Outra notação frequente para reunião disjunta é $\coprod$.

conjuntos envolvidos são mutuamente disjuntos: a escritura $\biguplus_{i=1}^{N} A_i$ subentende que os A_i são mutuamente disjuntos.

- Dados dois conjuntos A, B, definimos o produto cartesiano de A com B

$$A \times B = \{(a, b) : \ a \in A, \ b \in B\} :$$

é um novo conjunto cujos elementos são duplas ordenadas onde o primeiro elemento pertence a A e o segundo a B (note que $A \times B$ é um conjunto diferente de $B \times A$). Da mesma forma definimos o produto de mais conjuntos

$$A_1 \times A_2 \times ... \times A_n = \{(a_1, a_2, .., a_n) : \ a_i \in A_i : i = 1, ..n\} :$$

é formato por n-upla ordenadas cuja i-ésima entrada pertence ao i-ésimo conjunto do produto.

2.2.2 Multiconjuntos

Neste capítulo precisaremos de uma generalização do conceito de conjunto: definimos **multiconjunto**, uma coleção de objetos onde posso pegar cada objeto mais de uma vez (mas como no caso de conjuntos, não importa a ordem).

Dado um multiconjunto M, posso associá-lo a uma dupla (C, m) onde

- C é um conjunto
- $m : C \to \mathbb{N} \cup \{0\}$ é a *função multiplicidade de* M:
  $m(c) =$"o número de vezes que c está em M"

A **cardinalidade de um multiconjunto** (finito) é o número de seus elementos (contando repetições). Com a notação acima pode ser visto como a soma das multiplicidades

$$|M| = \sum_{c \in C} m(c) \, .$$

Exemplo 2.1. Sejam $A = \{1; 2; 3\}$ e $B = \{1; 2; 3; 1\}$, multiconjuntos. claramente $A \neq B$: de fato, $|A| = 3$ e $|B| = 4$.

O multiconjunto B pode ser associado à dupla (C, M) onde $C = \{1; 2; 3\}$ (conjunto!) e $m(1) = 2, \ m(2) = m(3) = 1$.

A associação não é única: poderíamos também associar a B a dupla composta por $C = \{1; 2; 3; 4\}$ e $m(1) = 2, \ m(2) = m(3) = 1, \ m(4) = 0$, e infinitas outras.

Cuidado em não confundir conjuntos e multiconjuntos: podemos dizer "sejam $A = \{1; 2; 3\}$ e $B = \{1; 2; 3; 1\}$, conjuntos", neste caso. $A = B$ e $|A| = |B| = 3$: o fato de repetir um elemento na descrição do conjunto não adiciona nada: para conjuntos, ou o elemento está ou não está. ★

2.3 Alguns princípios básicos

Nesta seção veremos alguns princípios bem simples (a prova é trivial) que servirão depois para contar os elementos de conjuntos mais complexos.

Teorema 2.2 (Princípio da adição)**.**
Se $\{A_i\}_{i=1,..,N}$ *é uma família (finita) de conjuntos FINITOS, mutuamente disjuntos, então*

$$\left|\bigcup_{i=1}^{N} A_i\right| = \sum_{i=1}^{N} |A_i|$$

Teorema 2.3 (Princípio do produto)**.**
Se $\{A_i\}_{i=1}^{N}$ *é uma família composta por N conjuntos FINITOS, todos de cardinalidade M e mutuamente disjuntos então*

$$\left|\bigcup_{i=1}^{N} A_i\right| = MN$$

Teorema 2.4 (Princípio da bijeção)**.**
Se A, B *são conjuntos FINITOS, então*
$|A| = |B|$ *se e só se existe uma bijeção* $b : A \to B$

Em palavras, o princípio da soma diz que reunindo conjuntos mutuamente disjuntos obtemos um conjunto com cardinalidade igual à soma das cardinalidades dos conjuntos reunidos, enquanto o princípio do produto simplesmente trata o caso em que os conjuntos reunidos são todos da mesma cardinalidade. O principio da bijeção nos fornece uma forma

de provar que dois conjuntos têm a mesma cardinalidade, construindo uma bijeção entre os dois.

Podemos reformular os princípios de várias maneiras.

Dizemos que $\{A_i\}_{i=1}^N$ é uma **partição** de um conjunto A se $A = \biguplus_{i=1}^N A_i$.
Cada conjunto A_i é dito **bloco da partição**. Usando esta definição, o princípio de adição pode ser reformulado em termos de partições da seguinte forma:

Teorema 2.5 (Princípio da adição - v2)**.**
Se $\{A_i\}_{i=1}^N$ é uma partição do conjunto FINITO A, então

$$|A| = \sum_{i=1}^{N} |A_i|$$

Uma versão muito útil do princípio do produto diz respeito a produtos cartesianos:

Teorema 2.6 (Princípio do produto - v3)**.**
Se $\{A_i\}_{i=1}^N$ é uma família de N conjuntos FINITOS, então[5]

$$|A_1 \times A_2 \times ... \times A_n| = |A_1| \cdot |A_2| \cdot ... \cdot |A_n|$$

$$\left|\prod_{i=1}^{N} A_i\right| = \prod_{i=1}^{N} |A_i|$$

Demonstração. Mostremos primeiro que $|A \times B| = |A| \cdot |B|$. É suficiente aplicar o Teorema 2.3 depois de perceber que $A \times B = \{(a, b) : \ a \in A, \ b \in B\}$ pode ser escrito como $\biguplus_{b \in B} \{(a, b) : \ a \in A\}$, que representa a reunião de $|B|$ conjuntos disjuntos cada um com $|A|$ elementos.

O caso com mais conjuntos pode ser feito por indução: se é verdade com $C_n = A_1 \times A_2 \times ... \times A_n$ então o caso $C_{n+1} = A_1 \times A_2 \times ... \times A_n \times A_{n+1} = C_n \times A_{n+1}$ é verdadeiro pelo que acabamos de provar para dois conjuntos. □

Veremos na Seção 2.4.2 mais uma formulação do princípio do produto.

[5] A notação $\prod$ indica um produto múltiplo, como $\sum$ para a soma. É usado tanto para o produto entre números como para o produto cartesiano de conjuntos.

2.3.1 O problema dos segmentos que unem N pontos no plano

Nesta Seção procuraremos resolver o seguinte problema:
"dados N pontos não alinhados no plano, quantos segmentos podemos desenhar usando eles como vértices?"

Observação 2.7. Nos problemas de contagem nem sempre é fácil adivinhar a estratégia que permite resolver um problema, por isso é útil conhecer várias técnicas. ◁

1^a solução:
Passo 1)

Resolvamos primeiro um problema mais fácil:

"Dados N pontos não alinhados no plano, quantos vetores (ou segmentos orientados) podemos desenhar usando eles como vértices?"

Neste caso precisamos, para cada possível vetor, selecionar o ponto de início e depois (entre os restantes) o ponto de fim.

A escolha do ponto inicial pode ser feita de N formas diferentes, a escolha do ponto de fim, apenas de $N-1$ formas diferentes, pois não posso usar o ponto inicial.

Também podemos pensar assim: se V é o conjunto dos N vértices, o problema de determinar o vetor é análogo ao problema de determinar as duplas ordenadas de elementos de V, onde o segundo elemento da dupla é diferente do primeiro, isto é, contar os elementos de

$$D := \bigcup_{a \in V} \{(a,b) : \ b \in V \setminus \{a\}\}.$$

O conjunto D está escrito como reunião disjunta de N conjuntos, cada um de cardinalidade $N-1$, logo pelo princípio do produto $|D| = N(N-1)$, e este será o numero de segmentos orientados. [6]

passo 2)

Agora que contamos os vetores, só falta perceber que a cada segmento correspondem os dois vetores que o percorrem nos dois sentidos possíveis, isto é, tem dois vetores para cada possível segmento.

Logo o número de segmentos será $N(N-1)/2$.

Veremos na Seção 2.6 uma forma de generalizar o raciocínio feito aqui.

2^a solução:
Fixemos um dos N pontos p_1. Claramente existem $N-1$ segmentos que têm ele por vértice (tenho $N-1$ possíveis segundos extremos).

Agora fixemos um outro ponto p_2: ainda existem $N-1$ segmentos que têm ele por vértice, ma um deles (o $\overline{p_1\,p_2}$) já foi contado, logo temos $N-2$ NOVOS segmentos.

Agora fixemos um outro ponto p_3... pensando como antes temos $N-3$ NOVOS segmentos tendo p_3 como vértice.

[6] Note que estamos usando o princípio da bijeção: construímos a bijeção que associa à dupla (a,b) o segmento que inicia em a e termina em b e então podemos dizer que os segmentos orientados são tantos quanto as duplas.

Continuando dessa forma, de p_i sairão $N-i$ NOVOS segmentos e a sequência concluirá ao tomar p_N do qual já não sai mais nenhum NOVO segmento.

Em conclusão, o número total de segmentos será a soma dos NOVOS segmentos de cada passo, isto é

$$\sum_{i=1}^{N}(N-i)\,.$$

Observe que mudando di índice $j = N - i$ a soma acima se torna

$$\sum_{j=0}^{N-1} j = N(N-1)/2\,,$$

como vimos no exemplo 1.15.

2.4 Algumas definições

Vejamos algumas definições típicas da combinatória, que usaremos durante o capítulo.

2.4.1 Listas, permutações e combinações

Chamamos **lista**, una sequência de objetos: neste caso importa quais são os objetos e também sua ordem. Temos também as seguintes definições.

- Uma **Lista de k objetos entre N**, é uma **sequências composta por k objetos, os quais podem ser escolhidos num conjunto de N elementos**.[7] Por exemplo:
  – 011010 é uma lista de 6 elementos em $\{0; 1\}$, isto é, uma lista de 6 elementos entre 2:
  – GATO, TOGA, TOTO,[8] são listas de 4 letras, ou seja, listas de 4 objetos entre 23 (as letras do alfabeto).

- Uma **permutação de k objetos entre N,** é uma lista (de k objetos entre N) na qual não é permitido repetir o mesmo objeto: por exemplo, nas listas do exemplo acima, GATO e TOGA são permutações, mas TOTO não é.

- Uma **permutação de N objetos** é o caso particular em que $k = N$: neste caso então todos os N elementos devem aparecer na permutação.

- Uma **Combinação simples de k objetos entre N,** é definida como um **subconjunto de k elementos, de um conjunto de N elementos**. Por ser um conjunto, não há uma ordem entre os elementos, por exemplo os subconjuntos do conjunto das letras $\{T, O, G, A\}$ e $\{G, A, T, O\}$ são idênticos, correspondem então à mesma combinação de 4 objetos entre as 23 letras.

[7] Às vezes dizemos lista de k objetos em um conjunto C, para indicar que os elementos podem ser escolhidos em C: se C é finito será então uma Lista de k objetos entre $\#C$.

[8] Como dissemos que importa a ordem, GATO e TOGA são duas listas diferentes!

- Uma **Combinação com repetição de k objetos entre N**

  é definida como um **multiconjunto de k elementos, escolhidos em um conjunto de N elementos**. Por ser um multiconjunto, não há uma ordem entre os elementos, por exemplo os multiconjuntos do conjunto das letras $\{D, A, D, O\}$ e $\{D, D, A, O\}$ são idênticos, mas diferem de $\{D, A, O\}$.

Observação 2.8. Uma lista de k objetos em C pode sempre ser vista como uma função

$$f : \{1; 2; ..; k\} \to C\,,$$

no sentido que a lista correspondente a f seria a sequência $f(1)\ f(2)\ ..\ f(k)$.

Nesta correspondência, as permutações correspondem às funções injetoras. No caso em que $k = \#C$, serão então as funções bijetoras. ◁

Lembrete:
Lista: importa ordem e pode repetir
Permutação: importa ordem e não pode repetir
Combinação simples: não importa ordem e não pode repetir
Combinação com repetição: não importa ordem e pode repetir

Como queremos aprender a contar quantas são todas as possíveis maneiras de formar os objetos definidos acima, fixemos a seguinte notação: indicaremos por

- L_k^N o número de listas de k objetos entre N,
- P_k^N o número de permutação de k objetos entre N,
- $P^N = P_N^N$ o caso particular do número de permutação de N objetos,
- C_k^N o número de combinações[9] simples de k objetos entre N,
- CR_k^N o número de combinações com repetição de k objetos entre N.

Veja no final do capítulo (Seção 2.9) o resumo dos resultados para estes valores.

Exemplo 2.9. Quantas são listas de k objetos entre N?

Já vimos isso ao contar os elementos do produto cartesiano: de fato, uma lista de k elementos em C é o mesmo que uma k-upla ordenada de elementos de C, isto é, um elemento de $C^k = \prod_{i=1}^k C$. Pelo princípio do produto $|C^k| = |C|^k$: se $|C| = N$ chegamos ao resultado

$$\boxed{L_k^N = N^k\,.} \tag{2.1}$$

★

[9] Cuidado: alguns livros usam a notação de ponta cabeça: C_N^k para o número de combinações de k objetos entre N. [SDK15a] usa $C(N, k)$. Não faça confusão entre as convenções.

Exemplo 2.10. Quantas são as permutações de 2 objetos entre N?

Já sabemos a resposta a isso: é suficiente usar o princípio da bijeção e perceber que o problema é análogo ao de determinar os segmentos orientados que unem N pontos.

Logo a resposta é

$$\boxed{P_2^N = N(N-1)\,.} \tag{2.2}$$

Quantas são as combinações de 2 objetos entre N?

Como antes podemos perceber que o problema é análogo ao de determinar os segmentos que unem N pontos (de fato agora só importa quais são os extremos, não sua ordem).

Logo a resposta é

$$\boxed{C_2^N = N(N-1)/2.} \tag{2.3}$$

★

Exercício 2.11. Quanta senhas de k dígitos podem ser formadas com N carácteres?
Quantas com ATÉ k dígitos?
Quantas com dígitos entre h e k?

★

Exemplo 2.12 (Aplicação em programação)**.** Quantas vezes o programa passa pela linha 8 (isto é, quanto vale "conta" no fim do programa)?

```
1  #include <stdio.h>
2
3  int main()    {
4      int i,j,conta=0,n=100;
5
6      for(i=1;i<=n;i++)
7              for(j=i+1;j<=n;j++)
8                      conta++;
9          printf("\n conta=%d",conta);
10 }
```

Podemos pensar assim: o programa passa exatamente uma vez para cada possível escolha de i, j satisfazendo $1 \leq i < j \leq n$. Quantas são?

Nenhum dos problemas vistos acima responde diretamente a esta pergunta. Vamos então usar o princípio da bijeção: associo a cada dupla i, j como acima, o conjunto $\{i, j\}$ com $1 \leq i, j \leq n$. É uma bijeção já que dado o conjunto $\{r, s\}$, como $r \neq s$ posso escolher o menor como i e o maior como j e formar uma (única) dupla como no programa.

Mas então a resposta é $C_2^n = n(n-1)/2$. De fato rodando o programa obtemos (com $n = 100$)

```
OUTPUT:

conta=4950
```

★

2.4.2 Uma versão mais geral do princípio do produto

Vamos provar e utilizar o seguinte Teorema.

Teorema 2.13 (Princípio do produto - v4)**.**
Seja S_n o conjunto das listas $(a_1, ..., a_n)$ de n elementos com as propriedades:

- *a_1 pode ser escolhido de h_1 maneiras,*
- *para todo $j : 1 \leq j < n$, escolhidos os elementos $a_1, .., a_j$, o elemento a_{j+1} pode ser escolhido de h_{j+1} maneiras.*

Então

$$|S_n| = \prod_{j=1}^{n} h_j \, .$$

Vejamos primeiro uma aplicação:
No exemplo 2.9 calculamos o número de listas de k elementos entre N já que percebemos que era o mesmo que contar os elementos de um produto cartesiano e logo utilizamos o Princípio do produto na versão do Teorema 2.6.

Para calcular o número de permutações de k elementos entre N precisamos uma forma diferente de pensar, já que as k-uplas do produto cartesiano podem ter elementos repetidos. O Teorema acima é o que precisamos:
– o primeiro elemento da permutação pode ser escolhido entre os N elementos (de N maneiras então),
– a escolha do segundo elemento é mais complicada, pois o conjunto das possíveis escolhas depende de qual foi a escolha do primeiro elemento. Porém, seja qual for a primeira escolha, o NUMERO de possibilidade será sempre $N - 1$,
– a cada passo então estamos na situação de ter um conjunto de possíveis escolhas dependendo das escolhas anteriores, mas o NUMERO de possibilidade será sempre $N - i$, onde i é o número dos termos já escolhidos. Em particular, no último passo já teremos escolhido k-1 elementos e teremos $N - (k - 1) = N - k + 1$ possibilidades de escolha.

O Teorema 2.13 diz então que o número de permutações de k elementos entre N será

$$\boxed{P_k^N = N(N-1)....(N-k+1).} \tag{2.4}$$

Observe que uma fórmula alternativa para P_k^N, usando fatoriais[10], é $\frac{N!}{(N-k)!}$.

Observação 2.14. No livro [SDK15a] é usada outra notação para este valor: $P_k^N = N^{\underline{k}}$ (k-ésima potência fatorial decrescente de N): o sentido do nome é que no lugar da k-ésima potencia de N (onde N é multiplicado por si mesmo k vezes), na k-ésima potência decrescente ele é multiplicado por $N - 1$, depois por $N - 2$, e assim por diante até um

total de k fatores. O livro define também a k-ésima potência fatorial crescente de N: $N^{\overline{k}} := N(N+1)(N+2)...(N+k-1)$ onde N é multiplicado por $N+1$, depois por $N+2$ e assim por diante até um total de k fatores. ◁

Note que no caso particular $k = N$, temos

$$\boxed{P_N^N = P^N = N!.} \tag{2.5}$$

Prova do Teorema 2.13. Se trata apenas de generalizar o raciocínio feito para o caso $n = 2$ na Seção 2.3.1, escrevendo o conjunto das listas como uma reunião disjunta.

A estrutura do teorema sugere usar indução.

- Se $n = 1$ é trivialmente verdade pois se trata apenas de escolher a_1 de h_1 maneiras.
- Suponha então que para certo k a afirmação seja verdadeira.
  Seja D o conjunto das k-upla escolhidas conforme as regras no teorema, e por hipótese de indução sabemos que $|D| = \prod_{j=1}^{k} h_j$.
  Agora construímos o conjunto das $k+1$-uplas adicionando a cada possível elemento de D o elemento a_{k+1}, pelo qual tem sempre h_{k+1} possibilidades. Então o conjunto E de todas as $k+1$-uplas será

$$E := \bigcup_{d \in D} \{(d, a_{k+1}) : \ a_{k+1} \text{ sendo uma das } h_{k+1} \text{ possibilidades}\}.$$

  Pelo princípio do produto do Teorema 2.3, esta reunião (disjunta) tem $|D| \cdot h_{k+1}$ elementos, isto é, $|E| = \prod_{j=1}^{k+1} h_j$.

Provamos então por indução o Teorema. Note que na escritura de E os possíveis valores de a_{k+1} dependem em geral do d, logo E não é um produto cartesiano, mas o fato que estes possíveis valores sejam sempre a mesma quantidade permite aplicar o Teorema 2.3. □

2.5 Relações

Vamos agora introduzir um assunto que será importante para aprender novas formas de contar elementos de certos conjuntos.

Definição 2.15.

[10] Dado $N \in \mathbb{N}$, $N!$ (N fatorial) é definido como o produto de todos os naturais até N: ex. $2! = 2$, $3! = 6$, $4! = 24$,...
Por convenção (e não só por isso) definimos também $0! = 1$. Note que isso é coerente com as formulas acima: $P_N^N = N! = \frac{N!}{(N-N)!}$.

- *Dados dois conjuntos X, Y, chamamos* **Relação de X em Y**: *um conjunto $R \subseteq X \times Y$.*
- *Se $X = Y$ chamamos de* **Relação em X**.
- *Quando $(x, y) \in R$ dizemos xRy: "x está em relação com y".*

Exemplos:

- Seja $X = \{1, 2, 3, 4\}$, $Y = \{a, b, c\}$. representando eles sobre duas retas como na figura 2.1, uma relação de X em Y é então um (qualquer) subconjunto de $X \times Y$ (o retículo verde na figura), por exemplo, o subconjunto dos pontos em vermelho define uma relação (a relação em que 2Ra, 2Rb, 3Rb e 4Rc).

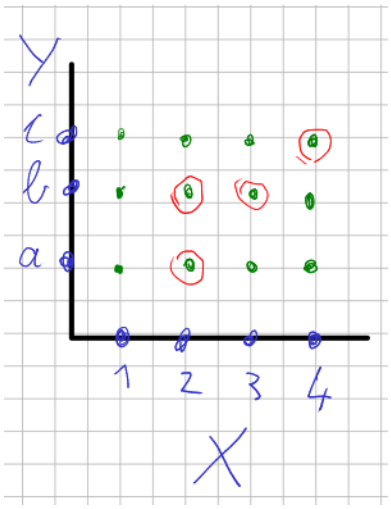
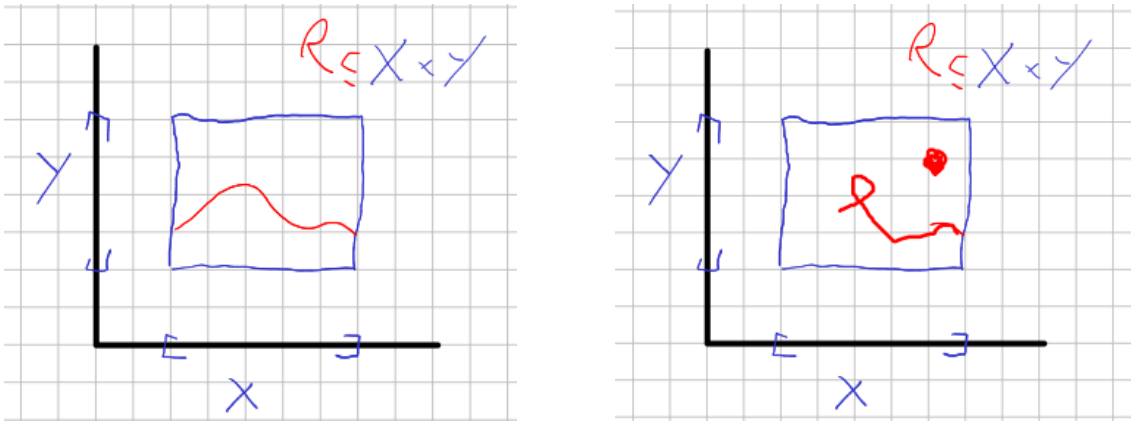


Figura 2.1: uma relação

- Sejam X, Y dois intervalos como nas figuras 2.2. Uma relação de X em Y é então um (qualquer) subconjunto de $X \times Y$ (o retângulo de azul nas figuras 2.2), por exemplo, o subconjunto dos pontos em vermelho, tanto no desenho 2.2a quanto no 2.2b.
- Dada uma função $f : X \to Y$, o conceito de Gráfico de F é definido como

$$G_f = \{(x, y) \in X \times Y : \ y = f(x)\}.$$

Logo, G_f é uma relação de X em Y: toda função pode ser associada a uma relação (seu gráfico), mas não vale o vice-versa, pois a função tem o requisito que para todo $x \in X$ deve existir exatamente um $y \in Y$ tal que $(x, y) \in G_f$. De fato, nos dois exemplos acima apenas a figura 2.2a representa uma relação que é também o gráfico de uma função. O caso 1 do exemplo abaixo também pode ser o gráfico de uma função.



(a) (b)

Figura 2.2

Exemplo 2.16. Vejamos mias exemplos:

1. R em $\mathbb{N}$: $xRy \iff x = y$ (identidade);
2. R em $\mathbb{N}$: $xRy \iff |x - y| = 1$;
3. R em $\mathbb{N}$: $xRy \iff x < y$;
4. R em $\mathbb{N}$: $xRy \iff x \leq y$;
5. R em X onde X é uma família de conjuntos: $ARB \iff A \subset B$;
6. R em X onde X é uma família de conjuntos: $ARB \iff A \subseteq B$;
7. R em $\mathcal{P}$ onde $\mathcal{P}$ é o conjunto das permutações de k elementos entre N:
   : $pRq \iff$ "p e q envolvem os mesmos k elementos".
   Por exemplo com $k = 2, N = 3$ temos $\mathcal{P} = \{12, 21, 23, 32, 13, 31\}$ e temos 12R21, 21R12, 12R12, 21R21,23R32, 32R23, 23R23.....

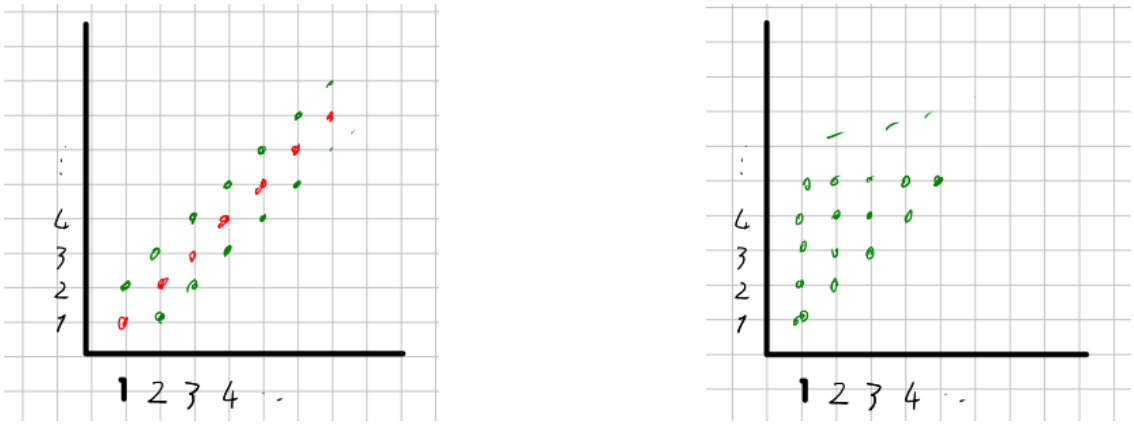
Na figura 2.3 estão (partes) dos gráficos de algumas das relações acima: em 2.3a em vermelho é a 1, em verde a 2, em 2.3b é a relação 4. ★

2.5.1 Propriedades das relações

Vejamos alguma propriedades que as relações podem ter.

Definição 2.17. *Uma Relação R em um conjunto X é dita*

- **Reflexiva**, *se $\forall x \in X$ vale xRx;*
- **Simétrica**, *se $xRy \iff yRx$;*



(a) (b)

Figura 2.3

- **Antisimétrica**, *se* $(xRy \wedge yRx) \implies (x = y)$;
- **Transitiva**, *se* $(xRy \wedge yRz) \implies (xRz)$. [11]

As primeiras três propriedades podem ser visualizadas no gráfico de R (por exemplo nas figuras 2.3): reflexiva significa que a diagonal está em R, simétrica significa que R é simétrico com respeito à diagonal, antisimétrica significa que se um ponto fora da diagonal está em R então seu simétrico não está.

No Exemplo 2.16 temos:

- são reflexivas as relações 1, 4, 6, 7;
- são simétricas as relações 1, 2, 7;
- são antisimétricas as relações 1, 3, 4, 5, 6;
- são transitivas as relações 1, 3, 4, 5, 6, 7.

Repare que a propriedade de simetria e de antisimetria não são uma a oposta da outra, de fato, 1 satisfaz ambas. Encontre um exemplo que não satisfaça nenhuma das duas.

Note que tanto 3 quanto 4 são antisimétricas. Para 4 isso é porque se $a \leq b$ e $b \leq a$ necessariamente $a = b$. Para 3 a verificação é menos óbvia: como $(a < b) \wedge (b < a)$ é sempre falsa, a implicação é sempre verdadeira independente de a e b serem iguais ou diferentes[12].

[11] Repare que temos quantificadores subentendidos nas três implicações: por exemplo, Antisimétrica significa que $\forall\ y \in X$ vale $(xRy \wedge yRx) \implies (x = y)$

[12] Quando uma implicação $a \implies b$ é verdadeira por ser sempre falsa a, dizemos que é "verdadeira por vacuidade".

2.5.2 Relações de equivalência

Uma Relação R em X é dita **relação de equivalência** se ela é **Reflexiva**, **Simétrica** e **Transitiva** Das relações do Exemplo 2.16, são relações de equivalência a 1 e a 7.

O teorema abaixo fornece uma propriedade fundamental das relações de equivalência.

Teorema 2.18. *Seja R em X uma relação de equivalência. Dado $x \in X$, seja*

$$C_x = \{z \in X : \ xRz\} \ .$$

Então vale:

1. *dados $x, y \in X$ vale $(C_x = C_y)$ XOR $(C_x \cap C_y = \emptyset)$,*
2. *a família*[13] $\{C_x\}_{x\in X}$ *é uma partição de X.*

Em palavras, o que diz o teorema é que a relação de equivalência divide o conjunto em um certo número de subconjuntos disjuntos, cada um dos quais contém os elementos que estão em relação entre si.

Estes subconjuntos que formam a partição $\{C_x\}_{x\in X}$ são ditos **classes de equivalência** (C_x é a **classe de equivalência de** x, também indicada por $[\![x]\!]$).

Exemplo 2.19. Considere a relação de equivalência 7 do Exemplo 2.16 com $k = 2$, $N = 3$. Então $C_{12} = \{12, 21\} = C_{21}$: as permutações em relação com 12 são a própria 12 e 21. Da mesma forma $C_{13} = \{13, 31\} = C_{31}$ e $C_{32} = \{32, 23\} = C_{23}$.

Logo a partição em classes de equivalência de $\mathcal{P}$ é

$$\mathcal{P} = \{12, 21\} \biguplus \{13, 31\} \biguplus \{23, 32\} \ :$$

são 3 classes cada uma de 2 elementos. ★

Exemplo 2.20. Outros exemplos importantes podem ser construídos da forma seguinte.

Considere R em $\mathbb{Z}$ definida como $xRy \iff x - y$ é par.
Então as classes de equivalência serão só dois: a dos pares e a dos ímpares (ambas contendo infinitos elementos).

Da mesma forma a relação definida por $xRy \iff$ "$x - y$ é múltiplo de p", define p classes de equivalência: por exemplo com $p = 3$ temos $\mathbb{Z} = C_1 \biguplus C_2 \biguplus C_3$ onde $C_1 = \{..., -5, -2, 1, 4, 7, ...\}$, $C_2 = \{..., -4, -1, 2, 5, 8, ...\}$, $C_3 = \{..., -6, -3, 0, 3, 6, ...\}$. ★

[13] Cuidado com esta notação. Como já comentamos no exemplo 2.1, quando descrevemos um conjunto podemos até repetir um elemento, mas isso não significa que ele esteja duas vezes no conjunto, assim, ao escrever $A = \{C_x\}_{x\in X}$ pode parecer que A tenha tantos elementos quanto X, mas na verdade pode ter menos, pois se $C_x = C_y$ ele corresponde a apenas um elemento de A.

Prova do Teorema 2.18. 1. Como $x \in C_x$ pela pr.reflexiva, ambas as afirmações não podem ser verdadeiras, pois se $C_x = C_y$ então $C_x \cap C_y$ não é vazio. Podemos então provar o XOR apenas mostrando que se $C_x \cap C_y \neq \emptyset$ então $C_x = C_y$.
Suponha então que $\exists\, z \in C_x \cap C_y$, isto é, xRz e yRz; pela pr.simetria vale zRy e pela transitividade vale xRy.
Agora, dado $w \in C_x$, vale wRx e usando xRy obtemos que wRy, isto é, $w \in C_y$. Como o mesmo raciocínio pode ser feito ao revés, obtemos que $C_x = C_y$ (todo elemento que está em um está no outro).

2. O fato que a família seja disjunta é consequência do ponto anterior, já que C_x e C_y ou são o mesmo elemento da família ou são disjuntos.
Só falta provar que a reunião das classes dá o X inteiro. Mas isso é óbvio já que $\forall x \in X$, $x \in C_x$ então está em uma das classes. □

Temos também o seguinte teorema, que mostra como qualquer partição pode ser vista como a partição em classes de equivalência de uma oportuna relação de equivalência.

Teorema 2.21. *Dada uma partição de um conjunto X ($X = \biguplus_{\alpha\in A} C_\alpha$) a Relação definida como*

$$xRy \iff x \text{ e } y \text{ estão no mesmo bloco da partição}$$

é uma Relação de equivalência.

2.5.3 Relações de ordem

Uma Relação R em X é dita

- **Relação de ordem (parcial)** se ela é **Reflexiva**, **Antisimétrica** e **Transitiva**.
- **Relação de ordem total** se ela é Relação de ordem (parcial) com a propriedade adicional que $\forall x, y \in X$ vale $(xRy \ \vee yRx)$.

Um conjunto X junto com uma relação de ordem R parcial (ou total) é dito **conjunto parcialmente (ou totalmente) ordenado**.

Exemplo 2.22. São relações de ordem a 4 ($\leq$) e a 6 ($\subseteq$) do exemplo[14] 2.16. Observe que a 3 ($<$) e a 5 ($\subset$) não são relações de ordem pois não são reflexivas[15].

[14] Também a 1 é uma reação de ordem (parcial), mas é um caso especial de pouco interesse, onde cada elemento está em relação apenas com si mesmo

[15] As relações 3 e a 5 tem propriedades parecidas às de ordem, mas com algumas deferências.

A 4 é uma relação de ordem total, enquanto (em geral[16]) 6 é apenas de ordem parcial, considere por exemplo $X = \{A, B, C\}$ com $A = \{1, 2\}$, $B = \{1, 2, 3\}$, $C = \{1, 3\}$. É claro que $A \subseteq B$, $C \subseteq B$, mas $A \subseteq C$ e $C \subseteq A$ são ambas falsas. ★

Definições

- Em um conjunto X ordenado com a relação R, chamamos **elemento mínimo**, um $x \in X$ (se existír) tal que xRy para todo $y \in X$.
- se X é totalmente ordenado e vale
  "todo $S \subseteq X$ com $S \neq \emptyset$ possui um elemento mínimo" dizemos que X **é bem ordenado**.

Exemplo 2.23. Com a relação "$\leq$", $\mathbb{N}$ é bem ordenado, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{Q}$ não são. Um intervalo em $\mathbb{R}$ também não é.
O conjunto $\{1/n : n \in \mathbb{N}\}$ não é bem ordenado mas $\{-1/n : n \in \mathbb{N}\}$ é bem ordenado! ★

2.6 Princípio da divisão e cálculo das combinações

O resultado abaixo é mais um princípio fundamental para resolver problemas de contagem.

Teorema 2.24 (Princípio da divisão)**.**
Se uma relação de equivalência divide um conjunto A de N elementos em k classes de equivalência, onde cada classe contém r elementos, então $k = N/r$.

Demonstração. A prova é consequência imediata do teorema do produto, de fato neste caso A é a reunião disjunta de k conjuntos com r elementos cada. □

Já usamos (intuitivamente) este princípio na Seção 2.3.1: primeiro contamos os vetores que unem os N pontos, depois definimos a relação de equivalência que agrupava os vetores com os mesmos extremos e identificamos as classes de equivalência com os segmentos que queríamos contar. Como cada classe contém dois elementos, chegamos a contar os segmentos dividindo por 2 o número de vetores.

Exercício 2.25. Use o raciocínio acima para contar os triângulos que podem ser desenhados usando N pontos não alinhados no plano. ★

[16] Se fosse $X = \{A, B\}$ então seria uma ordem total!

Para calcular o número C_k^N das combinações de k objetos entre N, vamos pensar de forma parecida. Seja C um conjunto com N elementos.

Primeiro consideramos o conjunto $\mathcal{P}$ das permutações de k objetos escolhidos em C; sabemos de (2.4) que $|\mathcal{P}| = P_k^N = N!/(N-k)!$.

Em seguida, consideramos a relação 7 do exemplo 2.16 em $\mathcal{P}$. Como é relação de equivalência, podemos aplicar a teoria da Seção 2.5.2. Em particular, a classe de equivalência C_p de uma permutação $p \in \mathcal{P}$ será composta de todas as permutações dos k elementos que compõe p, que sapemos ser $k!$.

Podemos associar a esta classe de equivalência o subconjunto de C que contém os k elementos de p. Desta forma, pelo princípio da bijeção, o número de combinações de k objetos entre N, será o número de classe de equivalência.

Como estamos na situação do Princípio da divisão (Teorema 2.24), em que um conjunto de P_k^N elementos é dividido em classes, cada uma composta por $k!$ elementos, concluímos que o número de classes de equivalência é $P_k^N/k!$.

Obtivemos

$$\boxed{C_k^N = \frac{N!}{k!\,(N-k)!}.} \tag{2.6}$$

Os números $\dfrac{N!}{k!\,(N-k)!}$ são chamados[17](veremos a motivação do nome na Seção 2.7) **coeficientes binomiais** e podem também ser indicados com a notação $\boxed{\binom{N}{k}}$.

Exemplo 2.26 (Aplicação em programação)**.** Quantas vezes o programa passa pela linha 9 (isto é, quanto vale "conta" no fim do programa)?

```
1  #include <stdio.h>
2
3  int main() {
4      int i,j,k,conta=0,n=10;
5
6          for(i=1;i<=n;i++)
7            for(j=i+1;j<=n;j++)
8              for(k=j+1;k<=n;k++)
9                  conta++;
10       printf("\n conta=%d",conta);
11 }
```

Pensando como no exemplo 2.12, o programa passa exatamente uma vez para cada possível escolha de i, j, k satisfazendo $1 \leq i < j < k \leq n$.

Como antes podemos associar a cada uma dessas triplas i, j, k, o conjunto $\{i, j, k\}$ com $1 \leq i, j, k \leq n$ (mostre que é uma bijeção).

Mas então a resposta é $C_3^n = \dfrac{n!}{3!(n-3)!}$. De fato rodando o programa obtemos (com $n = 10$)

[17] O livro [SDK15b] lê isso "n, k *a* k", entendendo que são as maneiras de agrupar k objetos entre n. Na versão [SDK11] usa "n, choose k": escolha k de n.

```
OUTPUT:

conta=120
```

★

2.6.1 Anagramas e rotulagem

Exercício 2.27. Quantos são os **anagramas** de GATO? e de CARACA?

Resposta:

Anagramas são de fato permutações. Porém precisamos tomar cuidado que, por exemplo, trocando entre elas duas 'A' de CARACA a palavra não muda. Logo a resposta para GATO é $P^4 = 4!$

Para CARACA podemos pensar assim: vejamos primeiro quantos seriam os anagramas se as letras fossem distintas (vamos numerar as letras iguais para distinguí-las: $C_1\, A_1\, R_1\, A_2\, C_2\, A_3$. Temos então $P^6 = 6!$ anagramas de $C_1\, A_1\, R_1\, A_2\, C_2\, A_3$.

Se agora definimos a relação de equivalência entre tais anagramas "só diferem pela ordem das A", obtemos classes de equivalências de $P^3 = 3!$ elementos cada, logo as classes são $6!/3!$. Se agora entre estas classes definimos a relação de equivalência "só diferem pela ordem das C", obtemos classes de equivalências (de classes!) de $P^2 = 2!$ elementos cada, logo as classes são $(6!/3!)/2!$.

Em geral, a resposta será que o número de anagramas é o fatorial do número total de letras dividido pelos fatoriais do número de letras de cada tipo (note que poderíamos pensar de dividir por $1! = 1$ por cada letra única e o resultado não mudaria). Em fórmulas, os anagramas de uma palavra composta por k_i letras L_i, $i = 1, .., T$ são

$$\boxed{\frac{(k_1 + k_2 + .. + K_T)!}{k_1!\, k_2! \ldots k_T!}}. \tag{2.7}$$

★

Exercício 2.28. **Problema de rotulagem.**

- Temos N garrafas (distintas), k rótulos verdes (idênticos) e $N - k$ rótulos azuis (idênticos). De quantas formas podemos rotular as N garrafas com os N rótulos?

  Sugestão: precisa escolher quais das N serão as verdes (ou quais das N serão as azuis, o resultado não muda!)

  O resultado é C_k^N $(= C_{N-k}^N)$.

- Temos N garrafas (distintas), k rótulos verdes (idênticos), h rótulos laranja (idênticos) e $N - k - h$ rótulos azuis (idênticos). De quantas formas podemos rotular as N garrafas com os N rótulos?

  Primeiro escolhemos quais das N serão as verdes. Para cada escolha, ainda precisamos escolher quais das $N - k$ restantes serão as laranja.

Logo, usando o princípio do produto do Teorema 2.13, o resultado é $C_k^N * C_h^{N-k}$, abrindo as contas isso é

$$\frac{N!}{k!\,h!\,(N-k-h)!}\,.$$

Outra forma de pensar, chegando ao mesmo resultado, é parecida à que usamos na hora de contar os anagramas: primeiro ordeno as garrafas ($N!$ permutações) dizendo que as primeiras k serão as verdes, as seguintes h as laranjas e as últimas as azuis.

Depois considero as classes de equivalência de permutações que só trocam entre elas garrafas verdes: cada classe tem $k!$ elementos, depois faço o mesmo com as laranjas e com as azuis, logo o valor de $N!$ deve ser dividido por $k!$, $h!$ e $(N-k-h)!$.

- Generalizando, Temos N garrafas (distintas), r cores de rótulos e $k_j : \ j = 1, .., r$ rótulos (idênticos) de cada cor, (com $\sum k_j = N$). De quantas formas podemos rotular as N garrafas com os N rótulos?

  A resposta é

$$\boxed{\frac{N!}{k_1!\,k_2!....k_r!}\,.} \tag{2.8}$$

Os números em (2.7) e (2.8) são ditos **coeficientes multinomiais**[18]: as notações usadas e a definição deles é a seguinte:

$$C^N_{k_1,k_2,..,k_r} = \binom{N}{k_1, k_2, .., k_r} = \frac{N!}{k_1!\,k_2!....k_r!}\,,$$

onde por convenção a soma dos k_j deve ser N ou a notação não faz sentido.[19] Pela fórmula, é claro que o coeficiente multinomial independe da ordem dos números k_j.

★

2.7 Coeficientes binomiais, propriedades, binômio e multinômio de Newton

2.7.1 Propriedades dos coeficientes binomiais

Como vimos na Seção 2.6, os números

$$C_k^N := \binom{N}{k} = \frac{N!}{(N-k)!\,k!} \qquad k, N \in \mathbb{Z}, \quad 0 \le k \le N.\ ^{20}$$

[18] Com $r = 3$ são também chamados trinomiais, Com $r = 4$ são também chamados quadrinomiais...

[19] Repare que a notação de coef. binomial e multinomial são diferentes e precisa ter cuidado em não confundir: a notação $\binom{5}{2}$ (coef. bin.) indica $\frac{5!}{2!\,3!}$, o mesmo número pode ser indicado, com a notação de coef. multinomial, $\binom{5}{2,3}$. Da mesma forma $C_2^5 = C_{2,3}^5$. Por outro lado, a escritura $C_{2,3}^8$ não faz sentido.

são ditos **Coeficientes binomiais**.

Vejamos alguns propriedades dos coeficientes binomiais. Vamos primeiro pôr eles em linhas contendo os C_k^N : $k = 0, .., N$:

N \ k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

Observando a tabela (chamada triângulo de Pascal) podemos enxergar algumas propriedades:

$$\binom{n}{k} = \binom{n}{n-k}, \tag{2.9}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \tag{2.10}$$

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n. \tag{2.11}$$

A propriedade (2.9) diz que toda linha é simétrica, (2.11) diz que a linha N tem soma 2^N, enquanto (2.10) diz que a soma de dois consecutivos é o número de baixo do segundo.

Note que apenas começando com o 1 da primeira linha[21], e interpretando como 0 todos os espaços vazios, podemos calcular todos os coef. binomiais apenas usando (2.10).

Todas estas propriedades podem ser provadas usando a definição de $\binom{N}{k}$ em termos de fatoriais. Porém é interessante ver o significado combinatório dessas propriedades, o que é também outra forma de prová-las.

(2.9) Ao escolher um subconjunto A de k elementos, de um conjunto U de N elementos, automaticamente fica definido o complementar A^c, com $N - k$ elementos.

Isso significa que podemos construir uma bijeção entre os subc. de k elementos e os de $N - k$ elementos, logo são a mesma quantidade.[22]

(2.11) Ao somar o número de todos os subc. de k elementos (com k de 0 a N) de um conjunto U com N elementos, obtemos TODOS os possíveis subconjuntos de U.

[20] Define-se também $C_k^N := \binom{N}{k} = 0$ para $k > N$ e para $k < 0$: isso não sai da formula com fatoriais mas expressa o fato que não tem como escolher k objetos de um conjunto com N se $N < k$ ou $k < 0$.

[21] Também podemos obter todos apenas preenchendo os $1 = C_0^N = C_N^N$ e obtendo os demais pela (2.10).

[22] Este é um exemplo do chamado princípio de simetria, que afirma que quando uma fórmula tem alguma simetria, em geral isso é a consequência de alguma propriedade do problema. Neste caso a propriedade é o fato de poder associar a cada conjunto o seu complementar.

Para contar eles, usamos outra bijeção: associo o conjunto $A \subseteq U$ à função $f : U \to \{0; 1\}$ definida como

$$f(u) = \begin{cases} 1 & se\ u \in A\,, \\ 0 & se\ u \notin A\,. \end{cases}$$

Logo o número total de subconjuntos é $2^{|U|}$ (veja a Observação 2.8 e o Exemplo 2.9).

(2.10) $\binom{N}{k}$ é o número de subc. com k elem de um U com N elementos.

Vamos fixar $v \in U$ e dividir estes subc. em dois grupos: os que contém v e os que não o contém:
os primeiros são $\binom{N-1}{k-1}$ pois já fixamos v, precisamos escolher os demais $k-1$ entre os $N-1$ que sobram,
os segundos são $\binom{N-1}{k}$ pois como v não está neles precisa escolher k elem. entre os $N-1$ que não são v.

2.7.2 Binômio de Newton

O nome de coeficientes binomiais é devido à aplicação na fórmula do **Binômio de Newton**, que expressa as potências do polinômio $(x+y)$: [23]

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}\,. \tag{2.12}$$

Tem várias formas de provar esta fórmula, por exemplo por indução sobre n. Aqui vamos ver como provar ela com um raciocínio combinatório.

Prova do binômio de Newton. Imagine abrir a potência $(x+y)(x+y)...(x+y)$: será formada por uma soma de monômios, cada um dos quais é construído escolhendo em cada parêntese ou x ou y. Logo tais monômios terão a forma $x^k y^{n-k}$, onde k indica o número de vezes que foi escolhido o x.

Porém existem $\binom{n}{k}$ maneiras diferentes de escolher os k fatores nos quais escolhemos x, por isso $x^k y^{n-k}$ aparecerá $\binom{n}{k}$ vezes na soma e então tem este coeficiente em (2.12). □

[23] Veja alguns casos simples:
$(x+y)^0 = \binom{0}{0}x^0y^0 = 1$
$(x+y)^1 = \binom{1}{0}x^0y^1 + \binom{1}{1}x^1y^0 = y + x$
$(x+y)^2 = \binom{2}{0}x^0y^2 + \binom{2}{1}x^1y^1\binom{2}{2}x^2y^0 = y^2 + 2xy + x^2$

2.7.3 Multinômio de Newton

Podemos também obter a fórmula do **Trinômio de Newton**:[24]

$$(x+y+z)^n = \sum_{(k_x,k_y,k_z)\in C} \binom{n}{k_x,k_y,k_z} x^{k_x} y^{k_y} z^{k_z}\,, \tag{2.13}$$

onde

$$C = \{(a,b,c) : a,b,c \geq 0 \text{ e } a+b+c = n\} \subseteq \mathbb{Z}^3\,.$$

A prova é análoga: imagine abrir a potência $(x+y+z)(x+y+z)...(x+y+z)$: será formada por uma soma de monômios, cada um dos quais é construído escolhendo em cada parêntese x ou y ou z. Logo tais monômios terão a forma $x^{k_x}y^{k_y}z^{k_z}$, onde agora $k_x,\ k_y,\ k_z$ devem ter soma n.

Esta vez, o número de maneiras diferentes de escolher os $k_x,\ k_y,\ k_z$ fatores nos quais escolhemos, respectivamente, x, y, z, pode ser calculado já que é equivalente ao problema da rotulagem do Exercício 2.28 (rotulamos cada fator com o rótulo que diz qual variável escolhemos nele), logo será mesmo $\binom{n}{k_x,k_y,k_z}$ como em (2.13).

Generalizando, obtemos a fórmula do **Multinômio de Newton**, que expressa a potencia n-ésima de uma soma de T variáveis, usando os coeficientes multinomiais:

$$(x_1 + x_2 + .. + x_T)^n = \sum_{(k_1,...,k_T)\in C} \binom{n}{k_1,..,k_T} x_1^{k_1}\, x_2^{k_2} \,...\, x_T^{k_T} \tag{2.14}$$

onde

$$C = \left\{(k_1,...,k_T) : k_1,...,k_T \geq 0 \text{ e } \sum_{i=1}^{T} k_i = n\right\} \subseteq \mathbb{Z}^T\,. \tag{2.15}$$

Veremos na Seção 2.33 uma forma de calcular $|C|$, isto é, o número de adendos que aparecem na somatória à direita de (2.14).

Exercício 2.29. Prove as fórmulas, usando o binômio/multinômio de Newton

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n\,, \tag{2.16}$$

$$\sum_{k=0}^{n} \binom{n}{k} (-1)^k = 0\,, \tag{2.17}$$

$$\sum_{a,b,c\geq 0:\ a+b+c=n} \binom{n}{a,b,c} = 3^n\,, \tag{2.18}$$

$$\sum_{a,b,c\geq 0:\ a+b+c=n} \binom{n}{a,b,c} (-1)^a = 1\,. \tag{2.19}$$

[24] Como exemplo,
$(x+y+z)^3 = (x^3+y^3+z^3)+3(x^2y+x^2z+y^2x+y^2z+z^2x+z^2y)+6xyz$, já que $\binom{3}{3,0,0} = 1$, $\binom{3}{2,1,0} = 3$, $\binom{3}{1,1,1} = 6$, e todas as possíveis formas (a menos de permutações) de escrever 3 como soma de 3 inteiros não negativos são $3+0+0,\ 2+1+0,\ 1+1+1$.

★

2.8 Combinações com repetição

Consideremos os seguintes três problemas

(AE) **Problema da arrumação da estante**: De quantas maneiras podemos dispor k livros (distintos) em N estantes (distintas)?

(AC) Variante: De quantas maneiras podemos dispor k livros (distintos) em N caixas (distintas)?

(AI) outra variante: e se os livros são idênticos?

Repare que a diferença está no fato que na estante temos uma ordem, então faz diferença pôr dois livros na mesma estante em ordem diferente, enquanto na caixa só importa quais livros estão nela. Esta diferença só vale para os livros distintos, pois se forem idênticos estante ou caixa não faz diferença, apenas importa quantos livros vão em cada uma delas.

- O problema (AC) na verdade pode ser modelado com um problema já visto: podemos definir a função f que diz, de cada livro, em qual caixa vai. Será então uma função de um conjunto de k elementos em um de N elementos, logo a resposta é N^k. [25]

- Para resolver o problema (AI) vamos usar o chamado "modelo bola-traço": pegamos k bolas ◯ e $N-1$ traços / e construímos uma sequência com eles, por exemplo

  ◯◯/◯//◯

  e interpretamos a sequência assim: cada traço separa uma caixa da seguinte, cada bola representa um livro na caixa correspondente. A sequência acima então representa uma distribuição de 4 livros em 4 caixas feita assim: 2 livros na caixa 1, 1 na 2, 0 na 3 e 1 na 4.

  Precisamos então contar de quantas formas podemos reordenar as bolas e traços, mas já sabemos isso, pois é como contar os anagramas[26] da "palavra" ◯◯/◯//◯ (ou em geral, da "palavra" com k ◯ e $N-1$ /), isto é

  $$\frac{(k+N-1)!}{k!(N-1)!} = \binom{k+N-1}{k} = C_k^{k+N-1}.$$

[25] Cuidado: não funcionaria pensar numa função que diz em cada caixa qual livro vai, pois dessa forma teria no máximo um livro por caixa e poderia repetir-se o livro: não seria o modelo certo para este problema.

- A este ponto fica fácil também resolver o problema (AE), de fato, repensando a como calculamos a fórmula para os anagramas, primeiro calculamos as permutações de todos os $k+N-1$ objetos, depois dividimos por $(N-1)!$ pois nada mudava em trocar a ordem dos traços, enfim dividimos por $k!$ já que também as bolas eram iguais. Se não fizermos a última divisão estaremos então distinguindo a ordem dos livros, resolvendo então o problema (AE), cuja resposta é

$$\frac{(k+N-1)!}{(N-1)!}.$$

Observação 2.30. Podemos resolver o problema (AE) por um argumento diferente (assim aprendemos também mais raciocínios úteis): considero N estantes,
• ao pôr o primeiro livro tenho N possibilidades (apenas escolher a estante);
• ao pôr o segundo livro, além de escolher a estante posso escolher de pôr o livro a direita ou a esquerda do único livro já posto, logo tenho $N+1$ possibilidades;
• ao pôr o terceiro terei $N+2$ possibilidade (não importa onde pus o segundo: se foi na mesma estante do primeiro então terei opção de pôr a direta ou a esquerda dos dois, ou no meio, se foi em outra estante então terei duas estantes onde posso escolher se vai a direita ou a esquerda).

Desta forma percebemos que o resultado final será $N(N+1)...(N+k-1)$ (k fatores crescentes começando de N). Este número é o mesmo que $\frac{(k+N-1)!}{(N-1)!}$ (verifique), e também já definimos ele na Observação 2.14 como k-ésima potencia fatorial crescente de N e indicamos por $N^{\overline{k}}$. ◁

Exercício 2.31. Podemos enfim resolver o problema das **combinações com repetição**: quanto vale CR_k^N?

Solução.
Vamos construir uma bijeção que nos leve a um dos problemas que acabamos de ver: lembre que CR_k^N é o número de multiconjuntos de k elementos, escolhidos em um conjunto A de N elementos (aqui, como podemos repetir elementos, k pode ser menor, igual ou até maior que N).

Vamos de novo usar o modelo bola traço, com k bolas (os k elementos do multiconjuntos) e $N-1$ traços, que vão separar N espaços que correspondem a cada um dos elementos de A: por exemplo, se $k=4$ e os elementos podem ser escolhidos no conjunto $A=\{1,2,3\}$ de cardinalidade $N=3$, então associamos à sequência

$$\bigcirc\,\bigcirc\,/\,\bigcirc\,/$$

o multiconjunto $\{1,1,2\}$ (o primeiro espaço correspondo ao elemento 1, que aparece então duas vezes, o segundo espaço ao 2, que aparece uma vez, o último espaço ao 3, que não aparece).

[26] Em vez de usar a analogia com anagramas podemos pensar de ter $k+N-1$ casinhas e precisamos escolher quais delas serão as bolas (as outras serão traços). A solução é então C_k^{N+k-1}: as maneiras de escolher as k casinha que serão bolas. Desta forma sai naturalmente a interpretação como combinação.

Concluímos que o número de multiconjuntos é igual ao número de sequências bola-traco, isto é,

$$\boxed{CR_k^N = C_k^{k+N-1}}. \tag{2.20}$$

★

Exemplo 2.32. Vamos considerar o conjunto $A = \{1, 2, 3\}$:
– os subconjuntos de dois elementos são apenas $C_2^3 = 3$: eles são $\{1, 2\}, \{1, 3\}, \{2, 3\}$;
– já os multiconjuntos de dois elementos escolhidos em A são $CR_2^3 = C_2^4 = 6$: eles são os três anteriores mais $\{1, 1\}, \{2, 2\}, \{3, 3\}$.

★

Exercício 2.33. Qual é a cardinalidade do conjunto em (2.15) ou, mais em geral,
quantas soluções tem o problema de determinar as V variáveis $x_1, ..., x_V \in \mathbb{Z}$ satisfazendo $x_i \geq 0$ e $\sum x_i = S$?

A resposta pode ser de novo obtida pelo modelo bola-traço: pego S bolas e $V - 1$ traços, e interpreto una sequência deles assim: cada um dos V espaços corresponde a uma das varáveis e o número de bolas que contém é o seu valor.

Por exemplo, com $V = 4$ e $S = 7$,

$$\bigcirc\bigcirc\bigcirc\bigcirc / \bigcirc\bigcirc // \bigcirc$$

corresponde à solução $x_1 = 4$, $x_2 = 2$, $x_3 = 0$, $x_4 = 1$.

Concluímos que o número de soluções é $C_S^{V+S-1} = CR_S^V$.

Em particular o conjunto em (2.15) tem CR_n^T elementos. ★

Exercício 2.34. Escreva o desenvolvimento de $(x + y + z)^3$ e verifique que possui $CR_3^3 = C_3^5 = 10$ termos.

Escreva o desenvolvimento de $(x + y + z + w)^2$ e verifique que possui $CR_2^4 = C_2^5 = 10$ termos. ★

Exercício 2.35.

1. Quantas soluções tem o problema de determinar as variáveis $x_1, ..., x_V \in \mathbb{Z}$ satisfazendo $x_i \geq 1$ e $\sum x_i = S$?
   Sugestão: troque de incognita: $z_i = x_i - 1 \geq 0$;

2. Quantas soluções tem o problema de determinar as variáveis $x_1, ..., x_V \in \mathbb{Z}$ satisfazendo $x_i \geq 0$ e $\sum x_i \leq S$?
   Sugestão: chame $s := \sum x_i$ e adicione uma incógnita $y := S - s$.

★

Exemplo 2.36 (Aplicação em programação)**.** Considere o programa do Exemplo 2.26, mas, nos dois for internos, inicialize $j = i$ e $k = j$ no lugar de somar 1. Quantas vezes o programa passa pela linha 9 (isto é, quanto vale "conta" no fim do programa)?

Agora o programa passa exatamente uma vez para cada possível escolha de i, j, k satisfazendo $1 \leq i \leq j \leq k \leq n$.

Agora podemos associar a cada tripla i, j, k como acima, o MULTIconjunto $\{i, j, k\}$ com $1 \leq i, j, k \leq n$, de fato, agora não tem mais o vínculo que i, j, k sejam diferentes: são admitidas repetições. (Mostre que é uma bijeção).

Então a resposta é $CR_3^n = \dfrac{(n+2)!}{3!(n-1)!}$. De fato rodando o programa obtemos (com $n = 10$)

```
OUTPUT:

conta=220
```

★

Exemplo 2.37 (**Contando funções**)**.** Os exemplos 2.26 e 2.36 podem ser lidos de forma diferente: no 2.26 determinamos o número de triplas estritamente ordenadas com valores em $\{1, 2, ..N\}$, o que equivale a contar as funções estritamente crescentes ($f(i) < f(i+1)$) de $\{1, 2, 3\}$ em $\{1, 2, ..N\}$. No 2.36 contamos as funções não decrescentes ($f(i) \leq f(i+1)$).

Generalizando, e juntando com o que já vimos, as funções $f : \{1, 2, .., k\} \to \{1, 2, ..N\}$ são:

- N^k em total,
- $N^{\underline{k}}$ injetoras (se $k \leq N$),
- C_k^N estr. crescentes (se $k \leq N$),
- CR_k^N não decrescentes.

★

2.9 Resumo fórmulas

- Número de *Listas de k objetos entre N* ($N \geq 1$, $k \geq 0$)[27]:

$$L_k^N := N^k \,.$$

- Número de *Permutações de k objetos entre N* ($0 \leq k \leq N$):

$$P_k^N := \frac{N!}{(N-k)!} = \prod_{j=N-k+1}^{N} j = N^{\underline{k}} \,.$$

- Número de *Permutações de N objetos* ($N \geq 0$):

$$P^N := P_N^N = N! \,.$$

- Número de *Combinações simples de k objetos entre N* ($0 \leq k \leq N$):

$$C_k^N := \binom{N}{k} = \frac{N!}{(N-k)!\,k!} \,.$$

[27] Nestas fórmulas, $N, k \in \mathbb{Z}$.
Lembrete: $0! = 1$

- Número de *Combinações com repetição de k objetos entre N* ($N \geq 1,\ k \geq 0$):

$$CR_k^N := \binom{k+N-1}{k} = C_k^{k+N-1}\,.$$

2.9.1 Mais fórmulas

- Número de *maneiras de rotular N objetos com N rótulos dos quais k_i rótulos de tipo i* (onde $i = 1, .., r$, $\sum_{i=1}^{r} k_i = N$)

$$C^N_{k_1,..,k_r} := \binom{N}{k_1, ..., k_r} = \frac{N!}{k_1! \dots k_r!}\,.$$

  É também o número de anagramas de uma palavra de N letras das quais k_i letras iguais de tipo i.

- Número de *maneiras de ordenar k objetos em N listas* (problema da arrumação da estante)

$$N^{\overline{k}} = \frac{N+k-1!}{(N-1)!} = \prod_{j=N}^{N+k-1} j\,.$$

- Número de *soluções do problema de determinar as variáveis $x_1, ..., x_V \in \mathbb{Z}$ satisfazendo $x_i \geq 0$ e $\sum x_i = S$*

$$CR_S^V = C_S^{V+S-1}\,.$$

Capítulo 3

Teoria dos números

Neste capítulo estudaremos alguns assuntos de Teoria do Números, que serão usados, no próximo capítulo, para o estudo do método de criptografia RSA.

3.1 Divisão de inteiros

Começamos definindo a noção de divisão de inteiros, diferente da que é geralmente usada para números reais. Ela está baseada no seguinte Teorema.

Teorema 3.1 (Teorema de divisão de Euclides)**.**
Dados $n \in \mathbb{Z}$ e $d \in \mathbb{N}$, existe uma única dupla $q \in \mathbb{Z}$, $r \in \{0, ..., d-1\}$ tal que

$$n = qd + r \tag{3.1}$$

Com a notação do Teorema podemos definir:

Definição 3.2. *Dados $n \in \mathbb{Z}$ e $d \in \mathbb{N}$, dizemos que a* **divisão entre inteiros** *n/d tem* **quociente q** *e* **resto r**, *(***d** *é dito* **divisor**).[1]

Exemplo 3.3. No sentido da divisão de inteiros, para calcular $3/2$ escrevemos $3 = q \cdot 2 + r$ e vemos que a única maneira de resolver isso com $r \in \{0, 1\}$ é $3 = 1 \cdot 2 + 1$.
Logo *a divisão de inteiros $3/2$ dá quociente 1 com resto 1.*
Analogamente,

[1] Em C, se n e d são integer (positivos), $q = n/d$, $r = n\%d$. Mas cuidado, se $n < 0$ a implementação no C retorna um resultado diferente da definição acima: de fato, pela definição dada, a divisão de inteiros $(-5)/3$ tem quociente -2 e resto 1, mas C retorna $(-5)/3 = -1$ e $(-5)\%3 = -2$.

• $2/3$ dá 0 com resto 2, pois $2 = 0 \cdot 3 + 2$;
• $13/5 = 2$ com resto 3, de fato $13 = 2 \cdot 5 + 3$;
• $(-13)/5 = (-3)$ com resto 2, de fato $-13 = (-3) \cdot 5 + 2$. ★

Uma motivação para este tipo de divisão pode ser a seguinte: se temos 7 bolos e queremos dividir entre 3 clientes, podemos proceder das seguintes formas:
– dividir um bolo em 3 e dar 2 bolos e $1/3$ para cada cliente (noção de divisão entre reais);
– dar apenas 2 bolos para cada cliente, sendo que vai sobrar um bolo invendido (noção de divisão de inteiros), o que é mais interessante se não queremos vender bolos cortados. Dependendo do problema prático uma ou outra noção de divisão será a mais indicada.

Definição 3.4. *Dados $n \in \mathbb{Z}$ e $d \in \mathbb{N}$, definimos* **n módulo d:**

$$n \ (mod \ d) := r \, ,$$

sendo r o dado pelo Teorema 3.1.

Para abreviar usaremos a notação[2] $nM_d = n \ (mod \ d)$.

Exemplo 3.5. Lembrando o exemplo anterior, temos
3 (mod 2)=1 (ou $3M_2 = 1$),
2 (mod 3)=2 (ou $2M_3 = 2$),
13 (mod 5)=3 (ou $13M_5 = 3$),
-13 (mod 5)=2 (ou $(-13)M_5 = 2$). ★

Prova do Teorema Teorema 3.1. Provamos por indução estrutural: considere a afirmação $A(i)$="o Teorema é verdadero se $(i-1)d \leq n < id$, ", sendo $i \in \mathbb{N}$.

CASO BASE: $A(0)$ é verdadeira pois se $0 \leq n < d$ então a escolha $r = n$, $q = 0$ satisfaz (3.1), além disso, pondo qualquer $q \neq 0$ em (3.1), $r = n - qd$ será negativo ou maior que $d - 1$, logo temos unicidade.

PASSO DE INDUÇÃO: suponha $A(j)$ verdadeira para certo $j \in \mathbb{N}$. Para provar $A(j+1)$ seja n tal que $(j+1-1)d \leq n < (j+1)d$; subtraindo d obtemos $(j-1)d \leq n - d < jd$, mas como $A(j)$ vale por hipótese de indução, sabemos que podemos escrever (de maneira única) $n - d = \widetilde{q}d + \widetilde{r}$ com $\widetilde{r} \in \{0, ..., d-1\}$. Mas agora re-somando d obtemos $n = (\widetilde{q}+1)d + \widetilde{r}$, logo a escolha $r = \widetilde{r}$, $q = \widetilde{q} + 1$ satisfaz (3.1) e também será única (ou re-subtraendo d contradiria a unicidade no caso $A(j)$).

Para provar o caso em que $n < 0$ podemos considerar
$\widetilde{A}$(i)="o Teorema é verdadeiro se $(-i)d \leq n < (-i+1)d$, ", sendo $i \in \mathbb{N}$, e provar pela mesma técnica. □

3.2 Aritmética módulo d

Vejamos algumas propriedades da operação de módulo.

[2]Esta notação é usada apenas nestas notas, não é uma notação comum, usem apenas neste curso ou depois de definí-la claramente.

Lema 3.6. *Dados $d \in \mathbb{N}$, $a, b \in \mathbb{Z}$,*

- $aM_d = (a + kd)M_d$ *para qualquer* $k \in \mathbb{Z}$,
- $(a + b)M_d = [(aM_d) + (bM_d)]\, M_d$,
- $(ab)M_d = [(aM_d)(bM_d)]\, M_d$.

Demonstração. Se a decomposição de a segundo o Teorema 3.1 é $a = qd + r$, então $aM_d = r$. Mas também $a + kd = (q + k)d + r$ e logo $(a + kd)M_d = r$.

Sejam $a = qd+r$ e $b = q'd+r'$ com $r = aM_d$ e $r' = bM_d$; logo $a+b = (q+q')d+r+r'$. Pegando módulo dos dois lados e usando a primeira propriedade obtemos o resultado $(a + b)M_d = [(q + q')d + r + r']M_d = (r + r')M_d$.

Da mesma forma, $ab = qq'd^2 + qr'd + q'rd + rr' = (qq'd + qr' + q'r)d + rr'$ logo $(ab)M_d = (rr')M_d$. □

Observação 3.7. Cuidado com os lados direitos das fórmulas do Lema 3.6: além de tomar os módulos dos dois termos ainda precisa tomar de novo depois de somar/multiplicar, isso porque a soma/produto poderia passad de $d - 1$.

Por exemplo:
– $(11+23)M_5 = 34M_5 = 4$, enquanto $11M_5+23M_5 = 1+3 = 4$, neste caso nem precisaria tomar módulo de novo,
– $(14+23)M_5 = 37M_5 = 2$, enquanto $14M_5+23M_5 = 4+3 = 7$, assim o resultado correto só é obtido tomando módulo de novo.

3.3 O conjunto $\mathbb{Z}_d$

Definição 3.8. *Seja $d \in \mathbb{N}$ e considere a relação de equivalência em $\mathbb{Z}$ (que já vimos no Exemplo 2.20):*

$$zRw \iff (z - w)(mod\ d) = 0\ :$$

vimos que $\mathbb{Z}$ fica dividido em d classes, cada uma composta por infinitos elementos; indicamos por $[z]_d$ a classe de equivalência de z. e por $\mathbb{Z}_d$ a família das classes de equivalência, isto é,

$$\mathbb{Z}_d := \{\ [z]_d\ \}_{z\in\mathbb{Z}}\ .$$

Observe que $\mathbb{Z}_d$ possui d elementos, por exemplo $\mathbb{Z}_3 = \{\ [0]_3, [1]_3, [2]_3\}\ .$

Observação 3.9. Cuidado em não confundir a classe com o módulo: dados, $n \in \mathbb{Z}$ e $d \in \mathbb{N}$ vale

$$n\ (mod\ d) = r\ \implies\ [n]_d = [r]_d\,,$$

mas não vale a implicação $\impliedby$, se não sabemos também que $r \in \{0, ..., d - 1\}$.

Repare que ao escrever a classe posso usar qualquer representante, isto é, $[3]_5 = [8]_5 = [-2]_5$. ◁

No conjunto $\mathbb{Z}_d$ podemos definir duas operações "$+_d$" e "$\cdot_d$" da seguinte forma:

- $[a]_d \ +_d \ [b]_d \ := [a+b]_d$, $\forall a, b \in \mathbb{Z}$ (**adição em $\mathbb{Z}_d$**),
- $[a]_d \ \cdot_d \ [b]_d \ := [ab]_d$, $\forall a, b \in \mathbb{Z}$ (**produto em $\mathbb{Z}_d$**).

Observação 3.10. Para poder fazer a definição acima é preciso mostrar que faz sentido. Em particular, precisamos verificar que ao pegar diferentes elementos $i \in [a]_d$ e $j \in [b]_d$ as classes $[i+j]_d$ e $[ij]_d$ são sempre as mesmas. Isso é garantido pelas propriedades do Lema 3.6. ◁

Observação 3.11. Sobre a notação:
- quando for claro o que é d vamos retirar da notação,
- quando for claro que estamos trabalhando em $\mathbb{Z}_d$ retiraremos também os colchetes:

Exemplo: fixado $d = 5$, no lugar de $[4]_5 +_5 [3]_5 = [7]_5 = [2]_5$
podemos escrever "$[4] + [3] = [7] = [2]$ em $\mathbb{Z}_5$"
ou também "$4 + 3 = 7 = 2$ no sentido de $\mathbb{Z}_5$". ◁

Vejamos algumas propriedades das oprações definidas.

Lema 3.12. *As operações $+_d$ e $\cdot_d$ definidas em $\mathbb{Z}_d$ satisfazem as propriedades comutativa, associativa, e distributiva do produto com respeito à adição.*

Além disso,
$[0]_d +_d \gamma = \gamma$, $\forall \gamma \in \mathbb{Z}_d$: *$[0]_d$ é elem. neutro da adição,*
$[1]_d \cdot_d \gamma = \gamma$, $\forall \gamma \in \mathbb{Z}_d$: *$[1]_d$ é elem. neutro do produto,*
$[a]_d +_d [d-a]_d = [0]_d$, $\forall a \in \mathbb{Z}$: *$[d-a]_d$ é o oposto de $[a]_d$.*

Demonstração. É suficiente usar a definição das operações e o fato que soma e produto em $\mathbb{Z}$ têm todas estas propriedades, por exemplo,
• $([a]+[b])+[c] = [a+b]+[c] = [(a+b)+c] = [a+(b+c)] = [a]+[b+c] = [a]+([b]+[c])$.
• Seja x um representante da classe γ, isto é, $\gamma = [x]_d$,
então $[1]_d \cdot_d \gamma = [1]_d \cdot_d [x]_d = [1x]_d = [x]_d = \gamma$.
Repare que também vale $[0]_d \cdot_d \gamma = [0]_d \cdot_d [x]_d = [0x]_d = [0]_d$. □

Exemplo 3.13. A definição das operações acima pode parecer muito abstrata, mas na verdade se aplica a muitos casos práticos.

Por exemplo, a "matemática do relógio" funciona como $\mathbb{Z}_{24}$ (ou $\mathbb{Z}_{12}$): se agora é 17hs, então daqui a 11 horas será $[11+17]_{24} = [4]_{24}$: 4 hs.

Podemos também enxergar $\mathbb{Z}_2$ como a matemática do par ou ímpar: $0 = [0]_2$ representa todos os pares e $1 = [1]_2$ representa todos os ímpares. As definições das operações nos dizem que $0+0 = 0\cdot 0 = 0\cdot 1 = 1+1 = 0$ enquanto $0+1 = 1\cdot 1 = 1$. ★

3.3.1 Uma definição alternativa

Em [SDK15a] é definido $Z_d := \{0, 1, ..., d-1\}$, com as operações

- $a +_d b := (a + b)M_d$, $\forall\, a, b \in Z_d$ (**adição em Z_d**),
- $a \cdot_d b := (ab)M_d$, $\forall\, a, b \in Z_d$ (**produto em Z_d**).

Esta é apenas uma forma diferente de fazer a definição, mas é obtida uma estrutura análoga à de $\mathbb{Z}_d$ (isto é, pode ser construída uma bijeção entre Z_d e $\mathbb{Z}_d$ que mantém as operações: a bijeção associa cada inteiro de Z_d com a sua classe de equivalência em $\mathbb{Z}_d$).

Completando a Observação 3.11, podemos dizer "$4+3 = 2$ no sentido de Z_5" (cuidado, $7 \notin Z_5$).

3.3.2 Definição de Anel e Corpo

Consederemos uma tripla $(\mathcal{K}, +, \cdot)$, formada por um conjunto $\mathcal{K}$ com uma operação $+$ dita *adição* e outra operação $\cdot$ dita *produto*.

Dizemos que $(\mathcal{K}, +, \cdot)$ é um **Anel comutativo com unidade** (ACCU) se valem as propriedades a seguir:

(A1) (associativa da adição) $(x + y) + w = x + (y + w)$, para quaisquer $x, y, w \in \mathcal{K}$;

(A2) (comutativa da adição) $x + y = y + x$, para quaisquer $x, y \in \mathcal{K}$;

(A3) (elemento neutro da adição) existe $z \in \mathcal{K}$ tal que $x + z = x$ para todo $x \in \mathcal{K}$;

(A4) (oposto da adição) para todo $x \in \mathcal{K}$ existe $y \in \mathcal{K}$ tal que $x + y = z$;

(P1) (associativa do produto) $(x \cdot y) \cdot w = x \cdot (y \cdot w)$, para quaisquer $x, y, w \in \mathcal{K}$;

(P2) (comutativa do produto) $x \cdot y = y \cdot x$, para quaisquer $x, y \in \mathcal{K}$;

(P3) (elemento neutro do produto) existe $u \in \mathcal{K}$ tal que $x \cdot u = x$ para todo $x \in \mathcal{K}$;

(D) (distributiva) $(x + y) \cdot w = x \cdot w + y \cdot w$, para quaisquer $x, y, w \in \mathcal{K}$.

Se além das propriedades acima, também vale

(P4) (inverso do produto) para todo $x \in \mathcal{K}$ com $x \neq z$, existe $y \in \mathcal{K}$ tal que $x \cdot y = u$,

então $(\mathcal{K}, +, \cdot)$ é dito **Corpo**.

Chamamos **invertível** um elemento $x \in \mathcal{K}$ pelo qual existe $y \in \mathcal{K}$ tal que $x \cdot y = u$: a particularidade dos corpos é então que todos os elementos exceto z (o neutro da adição) são invertíveis.

O Lema 3.12 nos diz que

$(\mathbb{Z}_d, +_d, \cdot_d)$ **é um anel comutativo com unidade.**

Exemplo 3.14. O exemplo mais conhecido de ACCU (que não é corpo) é $\mathbb{Z}$. Note que $\mathbb{N}$ não satisfaz (A4).
O conjunto das matrizes quadradas de um tamanho fixado (reais ou complexas) com as operações de soma e produto matricial forma um anel, mas não vale a comutatividade do produto (P2).
Exemplos de corpos são $\mathbb{R}$ e $\mathbb{Q}$. ★

As propriedades acima definem as noções de ACCU e de Corpo, mas existem outras propriedades que seguem delas, então sempre valem dentro dessas estruturas, mas não precisam ser incluídas na definição. Em particular temos:

- os neutros são únicos (os indicaremos com 0 e 1);
- oposto e inverso (se existir) de x são únicos (os indicaremos com $\overline{x}$ e x^{-1});
- $x \cdot 0 = 0 \quad \text{e} \quad \overline{x} = \overline{1} \cdot x$;
- $x \cdot w = 0$ implica x e/ou w não é invertível

Demonstração. Provamos a unicidade do neutro da adição.
Suponha que tanto $x + z = x \ \forall x \in \mathcal{K}$ quanto $x + z' = x \ \forall x \in \mathcal{K}$ (isto é, z e z' são ambos neutros da adição). Então $z = z + z' = z' + z = z'$, onde a primeira igualdade vem da propriedade de z', a segunda é a comutatividade e a última vem da propriedade de z. Logo $z = z'$, isto é, só pode ter um elemento neutro.
Provamos agora a última propriedade, chamada de regra de anulamento do produto.
Suponha $x \cdot w = 0$. Se x é invertível então multiplico dos dois lados por x^{-1} obtendo $x^{-1} \cdot x \cdot w = x^{-1} \cdot 0$ e logo $1 \cdot w = w = x^{-1} \cdot 0 = 0$.
Concluindo, mostramos que se x é invertível então $w = 0$ (e 0 não é invertível), logo certamente um dos dois não é invertível. □

Observação 3.15. Note que no caso de um corpo, o único elemento não invertível é 0, logo a última propriedade torna-se a conhecida propriedade $x \cdot w = 0 \Longrightarrow x = 0 \ e/ou \ w = 0$.
Se não estamos em um Corpo, isso pode não valer.
Por exemplo no anel das matrizes 2x2 temos

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

onde nenhum dos fatores é a matriz zero, mas ambos são matrizes não invertíveis.

No ACCU $\mathbb{Z}_6$ temos

$$[3] \cdot [4] = [12] = [0],$$

o que nos diz que $\mathbb{Z}_6$ não é corpo já que [3] e [4] não podem ser invertíveis em $\mathbb{Z}_6$. $\lhd$

Vejamos alguns exercícios sobre módulo e conjuntos $\mathbb{Z}_d$.

Exercício 3.16.

- Cálculo de 17452 (*mod* 9).

  Obviamente podemos calcular a divisão até obter o resto.

  Porém podemos usar as propriedades vistas aproveitado o fato que $[10]_9 = [1]_9$ e logo $[10^i]_9 = [10]_9^i = [1^i]_9 = [1]_9$ para qualquer $i \in \mathbb{N} \cup \{0\}$.
  Logo, em $\mathbb{Z}_9$,
  $[17452] = [1*10^5+7*10^4+4*10^3+5*10^2+2] = [1]+[7]+[4]+[5]+[2] = [19] = [1]$.
  Esta é a conhecida regra para o calculo de (*mod* 9).

- Cálculo de 17452 (*mod* 11):

  No caso do módulo 11 temos [10]=[-1] e logo $[10]^i = [(-1)^i]$, isto é, [1] ou [-1].
  Logo, em $\mathbb{Z}_{11}$,
  $[17452] = [1 * 10^5 + 7 * 10^4 + 4 * 10^3 + 5 * 10^2 + 2] = [1] + [-7] + [4] + [-5] + [2] = [19] = [-5] = [6]$.

- Cálculo de 5^{143} (*mod* 7).

  Vamos calcular, em $\mathbb{Z}_7$, $[5^j]$:
  $[5^1] = [5]$, $[5^2] = [25] = [4]$, $[5^3] = [4 * 5] = [-1]$, ...
  Já daqui percebemos que $[5^6] = [(-1)^2] = [1]$: o neutro do produto!
  A partir disso deduzimos que, para k inteiro, $[5^{6k+j}] = [5^j]$.
  Como $143 = 6 \cdot 23 + 5$ descobrimos que $[5^{143}] = [5^5] = [4(-1)] = [3]$.

  Observação: com esta técnica a conta ficou bem fácil. Atacando o problema diretamente seria até difícil com uma calculadora, pois 5^{143} poderia até dar overflow e logo dar um resultado completamente errado! $\lhd$

Exemplo 3.17. Vejamos algumas tabelas com os opostos e inversos em Z_d. Os inversos podem ser encontrados por tentatívas, visto o pequeno tamanho destes conjuntos se d é pequeno.

Em $\mathbb{Z}_6$

n	0	1	2	3	4	5
-n	0	5	4	3	2	1
n^{-1}	$\not\exists$	1	$\not\exists$	$\not\exists$	$\not\exists$	5

Em $\mathbb{Z}_3$

n	0	1	2
-n	0	2	1
n^{-1}	$\not\exists$	1	2

Em $\mathbb{Z}_9$

n	0	1	2	3	4	5	6	7	8
-n	0	8	7	6	5	4	3	2	1
n^{-1}	$\not\exists$	1	5	$\not\exists$	7	2	$\not\exists$	4	8

Em $\mathbb{Z}_7$

n	0	1	2	3	4	5	6
-n	0	6	5	4	3	2	1
n^{-1}	$\not\exists$	1	4	5	2	3	6

Das tabelas percebemos que $\mathbb{Z}_3$ e $\mathbb{Z}_7$ são corpos, enquanto $\mathbb{Z}_6$ e $\mathbb{Z}_9$ não são. ★

3.3.3 Subtração e divisão

Vimos que a adição e o produto estão definidas dentro da definição de ACCU e Corpo.

Subtração e divisão não são novas operações mas apenas as operações inversas das duas anteriores, que podem ser definidas quando existe o oposto (resp. inverso):

- a propriedade (A4) permite definir **subtração**: $x - y := x + \overline{y}$. [3]
- a propriedade (P4) (em corpos) permite definir **divisão** (exceto divisão por 0): $x : y := x \cdot y^{-1}$ (se $y \neq 0$);
- ainda podemos definir $x : y$ em um ACCU quando y é invertível.

Exemplo 3.18.

- Em $\mathbb{Z}^7$, $3 : 4 = 3 * (4^{-1}) = 3 * 2 = 6$ (de fato $[6 * 4] = [24] = [3]$).
- em $\mathbb{Z}^7$, $4 : 3 = 4 * (3^{-1}) = 4 * 5 = 6$ (de fato $[6 * 3] = [18] = [4]$).
- em $\mathbb{Z}^7$, $6 : 3 = 6 * (3^{-1}) = 6 * 5 = 2$ (de fato $[2 * 3] = [6]$).
- em $\mathbb{Z}^9$, $3 : 4 = 3 * (4^{-1}) = 3 * 7 = 3$ (de fato $[3 * 4] = [12] = [3]$).
- em $\mathbb{Z}^9$, $4 : 3$ não faz sentido, já que 3 não é invertível.

  Verifique que de fato não existe x tal que $x \cdot 3 = 4$ em $\mathbb{Z}^9$.
- em $\mathbb{Z}^9$, $6 : 3$ não faz sentido, já que 3 não é invertível.

  Verifique que neste caso existem três distintos $x \in \mathbb{Z}^9$ tais que $x \cdot 3 = 6$ em $\mathbb{Z}^9$: eles são $2, 5, 8$ (logo não dá para definir a operação de divisão já que teria um resultado indefinido).

Os últimos três casos mostram que a equação em $\mathbb{Z}^9$ $\quad ax = b \quad$ tem exatamente uma solução ($= a^{-1}b$) quando a é invertível, enquanto quando a não é invertível pode não ter solução, mas se tiver, então tem mais de uma.

Um fenômenos parecido é conhecido entre matrizes nxn: o problema $AX = B$ tem exatamente uma solução se A é invertível: $X = A^{-1}B$. Quando A não é invertível temos que, dependendo de B, ou não existe solução, ou existem infinitas soluções.

[3] De fato, $x + \overline{y} + y = x + 0 = x$, isto significa que $x + \overline{y}$ é aquele número que preciso somar a y para obter x, ou seja, a (única) solução "?" da equação $y+? = x$.

Veja o caso $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$: se $B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ não existem soluções de $AX = B$ (verifique), enquanto para $B = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ tem todas as soluções $X = \begin{bmatrix} 1 & 1 \\ x & y \end{bmatrix}$ com $x, y \in \mathbb{R}$. ★

3.3.4 Existência e cálculo de inverso

Na Seção anterior vimos que em um ACCU podem existir elementos invertíveis e não invertíveis. Também vimos que o cálculo do inverso em $\mathbb{Z}_d$ não é trivial. Nas próximas seções tentaremos responder às seguintes perguntas:

- quando $n \in \mathbb{Z}_d$ possui inverso?
- quando $\mathbb{Z}_d$ é um corpo?
- é difícil calcular o inverso de $n \in \mathbb{Z}_d$ (quando existe)?

Temos o seguinte primeiro resultado.

Lema 3.19. *Dados $d \in \mathbb{N}$ e $a \in \mathbb{Z}$, vale a equivalência:*

$$\exists\, [a]_d^{-1} \text{ (em } \mathbb{Z}_d) \iff \exists\, i, j \in \mathbb{Z} : \quad ia + jd = 1\,.$$

Quando existe, $[a]_d^{-1} = [i]_d$.

Demonstração. "$\Longrightarrow$" : por definição de inverso $[a]_d \cdot [a]_d^{-1} = [1]_d$. Seja $b \in [a]_d^{-1}$, então $[ab]_d = [1]_d$, o que significa que $ab = qd + 1$ para algum $q \in \mathbb{Z}$: pondo $i = b$ e $j = -q$ temos o resultado.

"$\Longleftarrow$" : se $ia + jd = 1$ então $[ia + jd]_d = [ia]_d = [i]_d \cdot [a]_d = [1]_d$, o que significa que $[i]_d$ é o $[a]_d^{-1}$.

□

Observação 3.20. Observe que mesmo que (como sabemos), o $[a]_d^{-1}$ seja único, as duplas i, j do Lema são infinitas, de fato se $ia + jd = 1$ então também $(i + kd)a + (j - ka)d = 1$ para qualquer $k \in \mathbb{Z}$. ◁

Observação 3.21. O Lema 3.19 não resolve diretamente o problema de obter o inverso de um elemento de $\mathbb{Z}_d$, apenas transforma o problema de determinar um elemento de $\mathbb{Z}_d$, num problema equivalente de determinar dois elementos de $\mathbb{Z}$. Aparentemente isso é ainda mais difícil (tem o duplo de incógnitas), mas veremos que ao contrário será mais fácil graças ao fato que as operações de $\mathbb{Z}$ tem melhores propriedades das de $\mathbb{Z}_d$ (veja na Observação 3.34). ◁

3.4 Números primos e fatoração

Vejamos nesta Seção algumas questões relacionadas com números primos.

Definição 3.22. *Dados, $n, d \in \mathbb{Z}$, dizemos que d divide n se existe $q \in \mathbb{Z}$ tal que $n = qd$ (outra possível definição, se $d \in \mathbb{N}$, é que $n \pmod d = 0$).*

- *$p \in \mathbb{Z} \setminus \{\pm 1\}$ é dito* **primo** *se os únicos $d \in \mathbb{Z}$ que dividem p são ± 1 e $\pm p$.*
- *$c \in \mathbb{Z} \setminus \{\pm 1\}$ é dito* **composto** *se não for primo.*[4]

Teorema 3.23 (Fatoração única)**.** *Todo $n \in \mathbb{N}$, $n \geq 2$ pode ser decomposto de forma única como*

$$n = \prod p_i^{e_i} \tag{3.2}$$

onde os p_i são primos (distintos) positivos e $e_i \in \mathbb{N}$.

Demonstração. A prova da existência já foi dada no Exemplo 1.16 por indução forte. A unicidade será mostrada na página 69. □

3.4.1 Algorítmo de fatoração

Uma forma elementar de encontrar a fatoração de um número n é a seguinte:
– começando de $d = 2$ (o menor primo) checamos se d divide n: se não dividir então tentamos com $d = 3, 5, ..$ até encontrar um divisor;
– encontrado um divisor aplicamos o mesmo algoritmo ao quociente obtido;
– se nenhum primo menor do número em análise o divide então ele é primo e terminamos a fatoração (na verdade já quando não encontramos divisores menores ou iguais da raiz do número podemos parar e deduzir que ele é primo).

Exemplo 3.24. Fatoração de 45.
– Começãndo por 45, seu menor divisor é 3,
– $45 : 3 = 15$, cujo menor divisor é ainda 3,
– $15 : 3 = 5$, cujo menor divisor é ele mesmo.
Logo $45 = 3^2 5$.

Costuma-se representar a conta acima da forma $\begin{array}{r|l} 45 & 3 \\ 15 & 3 \\ 5 & 5 \\ 1 & \end{array}$ onde o processo termina ao chegar a 1 e à direita aparecem todos os fatores primos repetidos conforme seu exponente.

Se não temos uma lista dos primos, podemos simplesmente tentar com todos os naturais começando de 2. ★

[4] ± 1 não são classificados nem como primos nem como compostos, para evitar que possamos pôr uma quantidade arbitrária deles dentro da fatoração de um número, perdendo a unicidade da fatoração.
Observe que 0 não divide nenhum $n \in \mathbb{Z}$ exceto ele mesmo. Por outro lado, qualquer inteiro divide 0.

O algoritmo acima é muito simples e pode ser implementado em C na forma abaixo.

```c
#include <stdio.h>

int minprimfac(int n)
{   int i;
    for(i=2;i<=n;i++)
        if(n%i==0) return i;
}

void printfactors(int n)
{   int m;
    m=n;
    do
    {   printf("\n%d",m=minprimfac(n));
        n=n/m;
    }while(n>1);
}
```

O output de printfactors(45) é

```
OUTPUT:
3
3
5
```

Infelizmente, o algoritmo acima deixa de ser útil para números muito grandes. Para ver o motivo vamos calcular o número de **operações necessárias** para descobrir que um número n de k dígitos (isto é, de tamanho comparável com 10^k) é primo.

Seguindo o algoritmo acima precisaremos testar a divisão $n : d$ para uma quantidade de tamanho 10^k de divisores. Melhorando o algoritmo de forma que pare ao chegar na raiz de n, ainda precisaríamos fazer uma quantidade de tamanho $10^{k/2}$ de divisões. Dispondo de uma tabela dos primos poderíamos reduzir ainda o número de operações, mas infelizmente o número de primos é tao grande que isso não traria uma vantagem considerável (veremos melhor na Seção 4.6).

Considerando que um computador normal realiza cerca de 10^{10} operações em vírgula móvel por segundo, tomando para excesso um valor de 10^{20} operações por segundo (superior ao de qualquer supercomputador), descobrimos que serão necessários $10^{k/2-20}$ segundos. Já para $k = 60$ isso ultrapassa o século! Com $k = 120$ ultrapassa a idade do universo.

Isso significa que a **fatoração de um número que contém fatores de muitos dígitos é inviável** com o algoritmo acima. Existem outros algoritmos mais eficientes mas nenhum permite fatorar números de centenas de dígitos em tempos razoáveis[5].

[5]Por exemplo existe o algoritmo de Fermat que consegue achar rapidamente eventuais fatores que estejam perto da raiz do número. Outros algoritmos funcionam em outros casos particulares, mas não existe uma técnica que funcione em geral.

3.5 MCD e algoritmo de Euclides

Definição 3.25. *Dados $a, b \in \mathbb{Z}$ definimos:* [6]

- **Máximo Comum Divisor de a e b** *(abrev. $MCD(a,b)$): é o maior $M \in \mathbb{N}$ que divide a e b.*
- *Se $MCD(a,b) = 1$ dizemos que a e b* **são relativamente primos** *(ou primos entre eles).*

Exercício 3.26. Uma forma elementar de encontrar o MCD é fatorar ambos os números e tomar todos os fatores comuns com a máxima potência possível. Por exemplo, para $MCD(225, 54)$ fazemos oseguinte:

225	3	54	2
75	3	27	3
15	5	9	3
5	5	3	3
1		1	

Deduzimos que o único fator comum é 3, que pode ser pego duas vezes, dando $MCD(225, 54) = 3^2 = 9$. ★

Infelizmente já sabemos que o método do exercício acima não é viável para números com fatores de muitos dígitos.

Procuramos então um método mais eficiente, que estará fundamentado no seguinte Lema.

Lema 3.27. *Dados $n, d, q, r \in \mathbb{Z}$ tais que $n = dq + r$,*

$$MCD(n, d) = MCD(d, r)\,.$$

Demonstração. Suponha que $k \in \mathbb{N}$ divida n e também d. Isto significa que existem $i, j \in \mathbb{Z}$ tais que $n = ki$ e $d = kj$.

Então $r = n - dq = (i - qj)k$ e logo k divide r também.

Da mesma forma se $k \in \mathbb{N}$ divide r e também d concluímos que k divide n também.

As duas afirmações significam que qualquer fator que divida d e n, divide também d e r e vice-versa, ou seja, as duas duplas tem os mesmos fatores em comum. Logo o MCD será igual. □

Em particular, o Lema 3.27 vale se $r = n \ (mod\ d)$. Baseado nisso, temos o **Algoritmo de Euclides** para o cálculo de $MCD(n_1, n_2)$ (com $n_1 > n_2 > 0$):

[6] Veja que pela definição se a é múltiplo de b então $MCD(a,b) = b$. Se $a = 0$, temos $MCD(0,b) = b$, exceto o caso $MCD(0,0)$ que não faz sentido pois qualquer número divide 0.

1) troque $MCD(n_1, n_2)$ por $MCD(n_2, n_3) = MCD(n_2, n_1 M_{n_2})$,
2) repita até chegar em $MCD(n_i, n_{i+1})$ com n_i múltiplo de n_{i+1},
3) conclua $MCD(n_1, n_2) = n_{i+1}$.

Veja que ao chegar ao passo final temos $n_i \% n_{i+1} = 0$, que pode ser usado como teste de parada. Repare também que o segundo termo do MCD é sempre estritamente menor do do passo anterior, logo o algoritmo deve necessariamente chegar ao fim.

Exemplo 3.28.
$MCD(225, 54)$: como 225=54*4+9, substituo por
$MCD(54, 9)$: como 54=9*6+0, concluo que
$MCD(225, 54) = 9$. ★

Uma implementação recursiva para o Algoritmo de Euclides (com alguns printf para ver o que acontece) é

```
#include <stdio.h>

int MCD(int n, int d) {
    printf("\n estou aqui com n=%d, d=%d, n(mod d)=%d",n,d,n%d);
    if(n%d==0) return(d);
    else return(MCD(d,n%d));
}
```

Para MCD(225,54) a função retorna 9, com

```
OUTPUT:
 estou aqui com n=225, d=54,  225=54*4+9
 estou aqui com n=54, d=9,  54=9*6+0
```

Para MCD(22504,345) a função retorna 1, com

```
OUTPUT:
 estou aqui com n=22504, d=345,  22504=345*65+79
 estou aqui com n=345, d=79,  345=79*4+29
 estou aqui com n=79, d=29,  79=29*2+21
 estou aqui com n=29, d=21,  29=21*1+8
 estou aqui com n=21, d=8,  21=8*2+5
 estou aqui com n=8, d=5,  8=5*1+3
 estou aqui com n=5, d=3,  5=3*1+2
 estou aqui com n=3, d=2,  3=2*1+1
 estou aqui com n=2, d=1,  2=1*2+0
```

Concluímos que 22504 e 345 são relativamente primos.

Para comparar com o algoritmo de fatoração, vamos calcular o número de operações necessárias no Algoritmo de Euclides, no exercício a seguir.

Exercício 3.29. Vamos estimar o número de **operações necessárias para descobrir o MCD** de números de k dígitos. Para isso, vamos analisar dois passos consecutivos do algoritmo:

$MCD(a,b) \to MCD(b,c) \to MCD(c,d)$.

Vamos provar que $d < b/2$:
– como $d = bM_c < c$ isso é verdade se $c \leq b/2$,
– em caso contrário, temos $2c > b > c$. Isso implica que ao escrever $b = qc + d$ necessariamente temos $q = 1$ e logo $d = b - c < b - b/2 = b/2$.

Isso significa que a cada dois passos o segundo termo do MCD é (pelo menos) dividido por dois; a cada 2h passos ele é (pelo menos) dividido por 2^h. Logo se $b < 2^h$ o algoritmo dura no máximo $2h$ passos.

Concluímos que uma estimativa do pior caso para o número de passos é $2log_2 b$, que corresponde aproximadamente a 6 vezes o número de dígitos de b. ★

Resumindo, para números de k dígitos, usando o algoritmo de fatoração podemos precisar de um número de operações do tamanho de $10^{k/2}$, enquanto usando o algoritmo de Euclides precisamos de um número de operações do tamanho de $6k$.

Tomando $k > 100$ isso significa que o MCD via fatoração é computacionalmente impossível enquanto via Euclides demora poucos segundos.

Veja o exemplo do cálculo de $MCD(22504, 345)$ na página 63: sendo $345 < 2^9$ esperaríamos no máximo 18 passagens. De fato, foram necessárias apenas 9 (veja que da primeira passagem para a terceira o divisor decresceu bem mais do esperado, indo de 345 para 29). Já a partir de $d = 8$ a descida foi mais lenta, mas sempre respeitando a regra obtida no Exercício 3.29.

3.5.1 Algoritmo de Euclides estendido

Analisando o algoritmo de Euclides deduzimos:

Lema 3.30. *Dados $n, d \in \mathbb{N}$, o $MCD(n,d)$ é combinação linear de n e d. Isto é, $\exists i, j \in \mathbb{Z}$: $MCD(n,d) = in + jd$.*

Demonstração. Cada passo do algorítmo tem a forma
"Troca $MCD(n_1, n_2)$ por $MCD(n_2, n_3) = MCD(n_2, n_1 M_{n_2})$",
onde por definição $n_1 = q_1 n_2 + n_3$ para um oportuno quociente $q_1 \in \mathbb{Z}$,
logo n_3 é combinação linear (CL) de n_1 e n_2.

Avançando com os passos, temos que n_j é CL de n_{j-2} e n_{j-1}.

Como o MCD é o último termo não nulo da sequência, ele é CL dos dois anteriores, que por sua vez são CL dos anteriores, e por consequência o MCD resulta ser CL dos dois termos iniciais da sequência. □

Os coeficientes i, j da combinação linear podem ser calculados junto com o MCD dentro do algoritmo, que neste caso é chamado **Algoritmo de Euclides estendido**.

Exemplo 3.31. Retomando o Exemplo 3.28, seguimos de novo as passagens.

Problema $MCD(225, 54)$: como

$$225 = 54 * 4 + \mathbf{9}, \tag{3.3}$$

substituo por $MCD(54, 9)$: como

$$54 = 9 * 6 + \mathbf{0}, \tag{3.4}$$

concluo que $MCD(225, 54) = 9$.

Agora explicitamos o $MCD = 9$ na penúltima equação (3.3) do algoritmo obtendo $MCD = 9 = 1 * 225 + (-4) * 54$: já obtivemos a CL linear procurada.

Vejamos um caso com mais passagens.

Problema $MCD(66, 14)$: como

$$66 = 14 * 4 + \mathbf{10}, \tag{3.5}$$

substituo por $MCD(14, 10)$: como

$$14 = 10 * 1 + \mathbf{4}, \tag{3.6}$$

substituo por $MCD(10, 4)$: como

$$10 = 4 * 2 + \mathbf{2}, \tag{3.7}$$

substituo por $MCD(4, 2)$: como

$$4 = 2 * 2 + \mathbf{0}, \tag{3.8}$$

concluo que $MCD(66, 14) = 2$.

Agora explicitamos o $MCD = 2$ na penúltima equação (3.7) do algoritmo obtendo $MCD = 2 = \mathbf{1 * 10 + (-2) * 4}$.

Agora explicitamos o 4 na equação anterior (3.6) obtendo $4 = 14 - 10$, que substituído no passo anterior nos dá
$MCD = 2 = 1 * 10 + (-2) * [14 - 10] = \mathbf{3 * 10 + (-2) * 14}$.

Agora explicitamos o 10 na equação (3.5) obtendo $10 = 66 - 4 * 14$, que substituído no passo anterior nos dá
$MCD = 2 = 3 * [66 - 4 * 14] + (-2) * 14 = \mathbf{(-14) * 14 + 3 * 66}$: obtivemos a CL linear procurada. ★

O procedimento mostrado no exemplo acima é útil para calcular a CL a mão em casos simples. Para implementar o Algoritmo de Euclides estendido vamos procurar como os coefficientes da CL mudam a cada passo.

Considere a genérica passagem do algoritmo que passa de $MCD(a, b)$ a $MCD(b, c)$ onde sabemos que $c = a - qb$ sendo $c = a\%b$ e $q = a/b$.

Sabemos pelos Lemas 3.27 e 3.30 que

$$MCD(a,b) = ia + jb = MCD(b,c) = Ib + Jc$$

para oportunos $i, j, I, J \in \mathbb{Z}$. Substituindo c por $a - qb$ obtemos $ia + jb = Ib + J(a - qb) = (I - qJ)b + Ja$ e deduzimos que

$$\boxed{j = I - qJ \quad e \quad i = J\,.} \tag{3.9}$$

Esta fórmula nos diz como calcular os coeficientes de um nível em função dos no nível seguinte. De fato, o único nível onde podemos saber os coeficientes é o passo final (veja como fizemos no Exemplo 3.31: primeiro fizemos todas as passagens para calcular o MCD e depois voltamos para trás nas passagens até chegar à combinação linear dos dois termos iniciais). Em particular, na passagem final temos $MCD(k\alpha, \alpha)$ onde α é exatamente o MCD buscado. Nesta passagem podemos então **inicializar a sequência** escrevendo $\alpha = 0 * (k\alpha) + 1 * \alpha$, isto é, pondo $\boxed{I = 0 \text{ e } J = 1}$ [7]. A partir disso e usando a fórmula (3.9) recursivamente obtemos os coeficientes i e j da CL que envolve os termos iniciais do problema.

Abaixo uma possível implementação do algoritmo (por comodidade i, j são definidas como variáveis externas, mas podem melhorar o algoritmo evitando isso), com vários printf que mostram as combinações intermédias.

Repare como a função é chamada recursivamente na linha 12 até quando o teste de parada na linha 6 inicializa os coeficientes i, j e inicia a volta. Ao longo da volta os coeficientes são recalculados pelas fórmulas (3.9) na linha 13, ficando com seu valor correto no final da recursão.

```
1  #include <stdio.h>
2  int i,j;
3  int MCD(int n, int d){
4      int tM,ti;
5      printf("\n estou aqui com n=%d, d=%d,  %d=%d*%d+%d",n,d,n,d,n/d,n%d);
6      if(n%d==0) {
7          i=0;j=1;
8        printf("\n voltando com n=%d, d=%d, %d=%d*%d+%d, i=%d, j=%d",n,d,n,d,
     n/d,n%d,i,j);
9        printf("; MCD(%d,%d)= %d: (%d)*%d+(%d)*%d=%d",n,d,d,i,n,j,d,d);
10       return(d);      }
11     else  {
12         tM=MCD(d,n%d);
13         ti=i;i=j;j=ti-j*(n/d);
14         printf("\n voltando com n=%d, d=%d, %d=%d*%d+%d, i=%d, j=%d",n,d,n,
     d,n/d,n%d,i,j);
15         printf("; MCD(%d,%d)= %d: (%d)*%d+(%d)*%d=%d",n,d,tM,i,n,j,d,tM);
16         return(tM);      }
17 }
```

[7] Lembre que na verdade existem infinitas possíveis combinações lineares para o MCD, de fato aqui poderíamos inicializar de infinitas outras formas, escolhendo $I \in \mathbb{Z}$ e pondo $J = 1 - kI$.

Para MCD(66,14) a função retorna 2, com

```
OUTPUT:
estou aqui com n=66, d=14,  66=14*4+10
 estou aqui com n=14, d=10,  14=10*1+4
 estou aqui com n=10, d=4,  10=4*2+2
 estou aqui com n=4, d=2,  4=2*2+0
 voltando com n=4, d=2, 4=2*2+0, i=0, j=1; MCD(4,2)= 2: (0)*4+(1)*2=2
 voltando com n=10, d=4, 10=4*2+2, i=1, j=-2; MCD(10,4)= 2: (1)*10+(-2)*4=2
 voltando com n=14, d=10, 14=10*1+4, i=-2, j=3; MCD(14,10)= 2: (-2)*14+(3)*10=2
 voltando com n=66, d=14, 66=14*4+10, i=3, j=-14; MCD(66,14)= 2: (3)*66+(-14)*14=2
```

Compare as passagens com as obtidas manualmente no Exemplo 3.31.

3.6 Ainda sobre existência e cálculo do inverso

Juntando resultados anteriores obtemos:

Teorema 3.32. *Dados $n, d \in \mathbb{N}$,*

$$MCD(n,d) = 1 \iff \exists\, i,j \in \mathbb{Z} :\ in + jd = 1$$

Demonstração. Pelo Lema 3.30 se $MCD(n,d) = 1$ então a combinação existe.

Viceversa, suponha que k seja um fator comum de n e d, isto é, $n = ka$ e $d = kb$, então $1 = ika + jkb$ e como são todos inteiros então k divide 1, logo $k = 1$. □

Corolário 3.33. *Dado $d \in \mathbb{N}$,*

- *dado $a \in \mathbb{Z}$,*

  $$[a]_d \text{ é invertível em } \mathbb{Z}_d \iff MCD(a,d) = 1.$$

  Além disso, $[a]_d^{-1}$ pode ser calculado pelo alg. de Euclides estendido.

- *$\mathbb{Z}_d$ é corpo $\iff$ d é primo.*

Demonstração. Basta combinar o Lema 3.19 com o Teorema acima. Como $MCD(p,n) = 1$ quando p é primo e $0 < n < p$, obtemos que todos os elementos não nulos de $\mathbb{Z}_p$ são invertíveis. □

O Corolário 3.33 confirma quanto observado no Exemplo 3.17, de fato 3 e 7 são primos, gerando os corpos $\mathbb{Z}_3$ e $\mathbb{Z}_7$, enquanto $\mathbb{Z}_6$ e $\mathbb{Z}_9$ não são corpos, em particular não são invertíveis todos seus elementos que tem o fator 3 (e/ou o fator 2 no caso de Z_6).

Além disso, o Corolário 3.33 nos diz que:

> para saber se existe o inverso precisamos calcular um MCD (que pode ser feito com o algoritmo de Euclides, até para números de muitos dígitos), e **para calcular o inverso precisamos da combinação linear, que também pode ser obtida usando o algoritmo estendido.**

Observação 3.34. Como dito na Observação 3.21, o cálculo do inverso tornou-se mais fácil na formulação alternativa de determinar as duas incógnitas $i, j \in \mathbb{Z}$ da CL, isso porque as duas incógnitas são determinadas, no Algoritmo de Euclides estendido, usando apenas as comuns operações de $\mathbb{Z}$, muito mais fáceis das operações de Z_d. ◁

Exercício 3.35. Calcule (se existir) o inverso de 42323 em $\mathbb{Z}_{323432}$.

Aplicando o algorítmo da página 66, para MCD(323432,42323) descobrimos, em instantes,

```
OUTPUT:
MCD(323432,42323)= 1: (2310)*323432+(-17653)*42323=1
```

Logo o inverso existe e é $[-17653]_{323432}$ (se queremos o resultado em Z_{323432} será $-17653+323432 = 305779$). ★

3.7 Ainda sobre primos

Lema 3.36. *Se $a, b, c \in \mathbb{N}$ com $MCD(a, b) = 1$ então*

- *(b divide ac) $\Longrightarrow$ (b divide c),*
- *(a divide c e b divide c) $\Longrightarrow$ (ab divide c),*

Se $a, b, p \in \mathbb{N}$ com p primo então

- *(p divide ab) $\Longrightarrow$ (p divide a ou b).*

Demonstração. Como $MCD(a, b) = 1$ sabemos que podemos escrever $ia + jb = 1$ para oportunos $i, j \in \mathbb{Z}$.
Logo $iac + jbc = c$. Agora,
– se b divide ac então b divide o lado esquerdo, e logo divide c;
– se a divide c e b divide c então podemos escrever $c = ma = nb$ com $m, n \in \mathbb{Z}$ e logo $ianb + jbma = c$, concluindo que ab divide c.

A última afirmação é consequência da primeira já que se p é primo e não divide nem a nem b então $MCD(p, a) = MCD(p, b) = 1$. □

Observação 3.37. Cuidado com a importância da hipótese $MCD(a, b) = 1$: sem ela as conclusões podem ser falsas, por exemplo:
– 4 divide $10 \cdot 18$ mas não divide nem 10 nem 18,
– 4 e 6 dividem 12 mas $4 \cdot 6$ não divide 12. ◁

Prova da unicidade da fatoração (Teorema 3.23). Supopnhamos por contradição que existam naturais com mais de uma fatoração, como $\mathbb{N}$ é bem ordenado seja n o menor deles.

Sejam $n = p_1p_2...p_k = q_1q_2...q_h$ duas fatorações em primos distintas de n (nesta escritura estamos repetindo eventuais fatores repetidos em vez de pôr exponente como na escritura (3.2)).

Como p_1 divide n, pelo último ponto do Lema 3.36, ele deve dividir uma dos q_i, digamos que divide q_1, isto é $p_1 = q_1$ já que são primos.

Então $n/p_1 = p_2...p_k = q_2...q_h$, mas então $n/p_1 < n$ é um natural que também possui duas fatorações distintas, contradizendo como foi tomado n. □

Exercício 3.38. Mostremos que **existem infinitos primos**.

Por contradição, se os primos fossem finitos, então existiria o maior deles, seja ele N.

Mostremos que $N! + 1$ é primo, o que será então uma contradição.

De fato, $N!$ é divisivel por cada natural menor ou igual a N, isto é, para cada natural $2 \leq j \leq N$ existe um natural k_j tal que $N!+1 = k_j \cdot j+1$, o que implica $N!+1 \ (mod\ j) = 1$, ou seja j não divide $N!+1$, que deve então ser primo, já que testamos com todos os primos existentes (se realmente fossem finitos).

3.8 Resultados importantes para o RSA

3.8.1 O Pequeno Teorema de Fermat

O Teorema abaixo é um resultado fundamental para a construção do método de criptografia RSA. Ele é chamado Pequeno Teorema de Fermat (o mais famoso Teorema de Fermat é o que envolve as soluções inteiras de $a^n + b^n = c^n$).

Teorema 3.39 (Pequeno Teorema de Fermat - PTdF)**.**
Se $p \in \mathbb{N}$ é primo então $a^{p-1}(mod\ p) = 1$ para todo $a \in Z_p \setminus \{0\}$.

Formulações equivalentes: Se $p \in \mathbb{N}$ é primo então

- $\gamma^{p-1} = [1]_p$ para todo $\gamma \in \mathbb{Z}_p \setminus \{[0]\}$.
- $a^{p-1}(mod\ p) = 1$ para todo $a \in \mathbb{Z}$ que não seja múltiplo de p.

Para a prova do Teorema 3.39 precisamos do seguinte Lema.

Lema 3.40. *Se $p \in \mathbb{N}$ é primo então,*
para todo $\gamma \in \mathbb{Z}_p \setminus \{[0]\}$, a lista

$$\gamma_i := i \cdot_p \gamma, \quad i \in \mathbb{Z}_p \setminus \{[0]\}$$

é uma permutação de $\mathbb{Z}_p \setminus \{[0]\}$.

Demonstração. Observe que o fato que p é primo implica que $\mathbb{Z}_p$ é corpo e logo todos seus elementos não nulos são invertíveis.
Multiplicando pelo inverso de γ deduzimos que

- $i \cdot_p \gamma = 0 \implies i = 0$,
- $i \cdot_p \gamma = j \cdot_p \gamma \implies i = j$:

a primeira diz que $\gamma_i \in \mathbb{Z}_p \setminus \{[0]\}$ para todo $i \in \mathbb{Z}_p \setminus \{[0]\}$,
a segunda diz que a aplicação $i \mapsto \gamma_i$, que agora pode ser vista com como uma mapa de $\mathbb{Z}_p \setminus \{[0]\}$ em si mesmo, é injetora, logo sobrejetora visto que a cardinalidade de domínio e contradomínio é igual e finita.

Isso prova a afirmação. □

Demonstração do Teorema 3.39. Mostremos a formulação em $\mathbb{Z}_p$. Consideremos o produto

$$P = \prod_{\forall\, i \in \mathbb{Z}_p \setminus \{[0]\}} i \cdot \gamma \,.$$

Pelo Lema

$$P = \prod_{\forall\, i \in \mathbb{Z}_p \setminus \{[0]\}} i \,, \tag{3.10}$$

já que permutando os elementos de $\mathbb{Z}_p \setminus \{[0]\}$ o produto não muda.

Por outro lado, pela associatividade,

$$P = \left(\prod_{\forall\, i \in \mathbb{Z}_p \setminus \{[0]\}} i \right) \cdot \left(\prod_{\forall\, i \in \mathbb{Z}_p \setminus \{[0]\}} \gamma \right) = \left(\prod_{\forall\, i \in \mathbb{Z}_p \setminus \{[0]\}} i \right) \cdot \gamma^{p-1} \,.$$

Igualando com a (3.10), como todos os elementos da produtória são invertíveis, multiplicando por seus inversos obtemos $\gamma^{p-1} = [1]_p$.

A última formulação segue das propriedades de módulo, já que
$a^{p-1}(mod\ p) = (a(mod\ p))^{p-1}$ e $a(mod\ p) \neq 0$ equivale a pedir que a não seja múltiplo de p.

□

Exercício 3.41. O cálculo de 5^{143} $(mod\ 7)$ do Exercício 3.16 agora poderia ser feito de maneira mais fácil, já que não precisamos fazer tentativas para descobrir que $[5]_7^6 = [1]_7$, mas é consequência do Teorema 3.39 (cuidado, isso não seria verdade se 7 não fosse primo: não podemos usar este atalho, por exemplo, para calcular 5^{143} $(mod\ 8)$). ★

Observação 3.42. O Teorema 3.39 nos fornece uma forma de calcular o inverso em $\mathbb{Z}_p$ com p primo, de fato $\gamma^{p-2} \cdot \gamma = \gamma^{p-1} = [1]$ e logo $\gamma^{-1} = \gamma^{p-2}$.
Isso é útil de um ponto de vista teórico, mas não prático se p é grande, já que o calculo de γ^{p-2} envolve $p - 3$ operações. ◁

3.8.2 O Teorema Chinês do resto

Outro resultado de Teoria dos Números que será necessário no desenvolvimento do RSA é o seguinte.

Teorema 3.43 (Teorema Chinês do resto - TCdR)**.** *Se $c, d \in \mathbb{N}$ são relativamente primos, então o sistema*

$$\begin{cases} x \ (mod\ c) = a \\ x \ (mod\ d) = b \end{cases}$$

com $a \in Z_c,\ b \in Z_d$, possui uma única solução $x \in Z_{cd}$.
A solução é $x = (bic + ajd) \ (mod\ cd)$ onde $ic + jd = 1$.

Demonstração. Consideremos a função $f : Z_{cd} \to Z_c \times Z_d : x \mapsto (x \ (mod\ c), x \ (mod\ d))$.

Como a cardinalidade (finita) de domínio e contradomínio é igual a cd, se mostrarmos que f é injetora então será também bijetora e isso prova a existência e unicidade da solução.

Suponha $f(x) = f(y)$, então

$$\begin{cases} x \ (mod\ c) = y \ (mod\ c) \\ x \ (mod\ d) = y \ (mod\ d) \end{cases}, \quad \Longrightarrow \quad \begin{cases} x - y \ (mod\ c) = 0 \\ x - y \ (mod\ d) = 0 \end{cases}.$$

Mas isso significa que c e d ambos dividem $x - y$ e pelo segundo ponto do Lema 3.36 (por serem primos entre eles) deve valer que cd dividem $x - y$.

Mas como $x, y \in Z_{cd}$, temos que $-cd < x - y < cd$ e logo $x - y = 0$. Isso mostra que $x = y$ e f é bijetora.

Para verificar a fórmula é só tomar módulo de $x = bic + ajd$:
$x \ (mod\ c) = (bic + ajd) \ (mod\ c) = ajd \ (mod\ c) = a(1 - ic) \ (mod\ c) = a \ (mod\ c) = a$,
e o mesmo com o módulo d.
Isso mostra que $bic + ajd$ sempre satisfaz o sistema, logo $(bic + ajd) \ (mod\ cd)$ é a única solução em Z_{cd}. □

Observação 3.44. O Teorema acima pode ser estendido a sistemas de mais de duas equações, pedindo que todos os módulos envolvidos sejam a dois a dois primos entre eles. ◁

Capítulo 4

RSA

4.1 Criptografia

Neste capítulo estudaremos alguns conceitos de criptografia.

Tipicamente o problema da criptografia é conseguir enviar uma mensagem que só possa ser lida pelo destinatário e não para outros que possam interceptar a mensagem.

Um simples exemplo clássico é o chamado **código de César**, usado pelo imperador romano para se comunicar com os comandantes do exército. O código de César consiste em trocar as letras de uma forma pre-concordada. Uma versão ainda mais simples é apenas concordar uma translação no alfabeto, por exemplo

$$a \to d,\ b \to e,\ c \to f, ...,\ w \to z,\ x \to a,\ y \to b,\ z \to c\,. \tag{4.1}$$

Este é um exemplo de **Criptografia de chave secreta:** quando remetente e destinatário concordam um código. Este tipo de criptografia tem algumas **desvantagens.** A principal é que **os dois envolvidos precisam conhecer o código**. Isso implica então que

- o código pode ser roubado mais facilmente,
- o código precisa ser concordado com antecedência ou enviado separadamente.

Além disso, o código de César tem a desvantagem de ser muito fácil de quebrar até sem precisar roubar o secreto.

Exemplo 4.1. O que está escrito?
D OLJHLUD UDSRVD PDUURP VDOWRX VREUH R FDFKRUUR FDQVDGR‡

Se apenas soubermos que a mensagem é em português, é razoável pensar que as letras sozinhas (D e R) serão "a" e "o" . Se o código é apenas de translação então isso já nos diz que será $D = a$ e $R = o$ e o código está quebrado.

Mesmo se for uma permutação mais geral do alfabeto, podemos observar que U será uma das poucas letras que podem aparecer em dupla ("r" ou "s"), as demais vocais podem

ser encontradas pela alta frequência, e uma vez feito isso será fácil completar a mensagem. ★

Até mesmo versões mais complicadas do código de César, por exemplo acoplando as letras de 2 em 2 e fazendo uma permutação das 24^2 duplas, podem facilmente ser quebradas com técnicas deste tipo e a ajuda de um computador.
De fato, se a mensagem for comprida o suficiente, é possível identificar o idioma apenas comparando a frequência relativa com que aparecem as letras (ou as duplas). Uma vez conhecido o idioma é fácil identificar cada letra pela frequência com que aparece.

Um tipo de criptografia mais interessante é a **Criptografia de chave pública:** quando o algoritmo de codificação é público, mas o de decodificação é conhecido apenas pelo destinatário (ou vice-versa).

Por exemplo, um método assim poderia ser usado pelos comandantes do exército romano para informar César da posição das tropas: ninguém poderia ler a mensagem interceptada, nem mesmo se capturar o comandante e descobrir o algoritmo de codificação, mas apenas César saberia ler a mensagem.

4.2 Ideia geral da criptografia de chave pública

Na criptografia de chave pública temos:

- uma função de codificação **publica** P, que codifica uma mensagem m em uma mensagem $c = P(m)$;
- uma função de decodificação **secreta** S, que seja a inversa da P, isto é, $P(S(m)) = m$ e $S(P(m)) = m$.

Além disso,

> **P e S devem ser tais que mesmo quem conhece P não consegue descobrir S.**[1]

Dadas as funções P e S, elas podem ser usadas de várias formas.

- **Envio de texto criptografado:** qualquer um pode enviar $c = P(m)$, mas o dono do secreto é o único que pode ler $m = S(c) = S(P(m))$. [2]

[1] Este requisito pode parecer estranho, será esclarecido na Seção 4.2.1.

[2] Um exemplo pratico é o seguinte: os clientes do banco precisam enviar sua senha secreta para confirmar suas compras. Para isso a encriptam com a função P (que é conhecida, por exemplo publicada no site do banco). Mesmo se alguém interceptar a mensagem, não descobrirá a senha pois apenas o banco conhece S e pode decriptá-la.

- **Envio de texto assinado:** o dono do secreto pode enviar $a = S(m)$, qualquer um pode ler $m = P(a) = P(S(m))$ e terá a certeza que quem enviou foi o dono do secreto.[3] [4]

- **Envio de texto criptografado e assinado:** Para isso precisamos que cada interlocutor seja dono de um secreto: sejam
  - P_A e S_A as chaves pública e secreta de A.
  - P_B e S_B as chaves pública e secreta de B.

  Se A envia $x = S_A(P_B(m))$ todos podem obter $P_B(m)$ $(= P_A(x))$, mas só B pode decriptar calculando $m = S_B(P_B(m))$ e saberá que foi A quem enviou.

  Uma outra forma, equivalente, de fazer isso é assim: se A envia $y = P_B(S_A(m))$ só B pode obter $S_A(m)$ $(= S_B(y))$, da qual ele obtém $m = P_A(S_A(m))$ e terá certeza que foi A quem enviou.

  Veremos na Observação 4.7 que na verdade apenas uma das duas formas descritas poderá ser usada.

4.2.1 Requisitos para as funções S e P

Vejamos quais devem ser as propriedades de S e P para obter um método de criptografia com as propriedades desejadas.

- Queremos que P e S sejam dadas por uma fórmula analítica, de forma que possam ser facilmente aplicadas, em particular, será interessante que a P seja suficientemente leve para ser calculada com computadores comuns, enquanto S poderia requerer uma potência computacional maior (em geral serão entidades como bancos ou nações que precisarão aplicar S).

- O requisito que mencionamos "quem conhece P não consegue descobrir S", de um ponto di vista teórico não faz sentido, pois dada uma função invertível a sua lei define univocamente a lei da sua inversa. Precisamos então entender este requisito da seguinte forma.

  (A) Não pode ser computacionalmente viável, dado c, descobrir $m = S(c)$ calculando $P(i)$ até encontrar o resultado c, nem pre-compilar uma tabela com todos os $P(i)$ para depois buscar o c nela.

[3] Para entender esta afirmação, precisamos pensar (será esclarecido em na Seção 4.3.1) que ao calcular $m = P(a)$ o resultado não terá algum sentido a não ser que a tenha mesmo sido produzido como $S(m)$, por isso, ao obter algo significativo usando a função P o recebente terá a certeza que a mensagem foi produzida usando a S.

[4] Um exemplo pratico é o seguinte: os clientes do banco precisam receber o endereço correto para acessar o banco, isso não precisa ser criptografado pois não é um segredo, mas o cliente precisa ter certeza de não estar sendo encaminhado para un site pirata que irá roubar seus dados, então precisa ter certeza que o que recebe seja mesmo enviado pelo banco. Para isso o banco assina a mensagem com sua função S e o cliente poderá ler usando a P.

(B) já que P é dada por uma fórmula analítica, não deve ser possível deduzir dessa fórmula a fórmula de S (por exemplo, se fosse $c = P(m) = e^m$ poderíamos obter $m = S(c) = \ln(c)$).

O requisito ((A)) implica que P e S devem **agir sobre conjuntos "grandes"**, por exemplo se seu domínio tiver 10^{100} elementos, uma tabela dos resultados ocuparia mais memória da de todos os computadores do mundo e um tempo inviável para ser calculada. [5]

O requisito ((B)) ficará claro mais tarde (veja a Observação 4.5).

Observação 4.2. Por causa do requisito ((A)) poderia se pensar que seria melhor trabalhar com conjuntos infinitos, como $\mathbb{R}$ ou $\mathbb{Z}$. Na verdade esta não é uma boa ideia em vista do requisito ((B)), porque o conjunto $\mathbb{R}$ possui muitas propriedades "boas" (topologia, ordem, completeza, ...) que ajudam na hora de inverter uma função, e $\mathbb{Z}$ é um subconjunto de $\mathbb{R}$ cujas operações são apenas a restrição a $\mathbb{Z}$ das operações de $\mathbb{R}$.

De fato, dada uma função contínua $f : \mathbb{R} \to \mathbb{R}$ bijetora, a inversa pode sempre ser pelo menos aproximada: dado $y = f(x)$ podemos aproximar x tao bem quanto quisermos aplicando o método das secantes ou de bisseção (se tivermos a derivada de f podemos até usar o método de Newton), que apenas precisam saber calcular valores de f pare serem aplicados.

Analogamente, dada uma função $\widehat{f} : \mathbb{Z} \to \mathbb{Z}$ bijetora, podemos interpolar ela com uma função $f : \mathbb{R} \to \mathbb{R}$; dado $y = \widehat{f}(x) = f(x)$, podemos aproximar x como descrito acima e uma vez que a aproximação for boa o suficiente para identificar apenas um inteiro, teremos obtido o valor exato de x.

Por causa disso, será preferível trabalhar nos conjuntos Z_d com d muito grande, que possuem operações que servirão para definir S e P por meio de uma fórmula, mas suficientes elementos para o requisito ((A)). Por outro lado, a falta de muitas propriedades das operações de Z_d (que NÃO são são apenas a restrição a Z_d das operações de $\mathbb{R}$) permitirá satisfazer também o requisito ((B)). ◁

Exemplo 4.3. Abaixo temos duas formas de construir funções bijetoras em Z_d definidas por fórmulas fáceis de calcular.

1. Dado $a \in \mathbb{Z}$, definimos
$$P : Z_d \to Z_d : m \mapsto P(m) = (m + a)(mod\ d)\,.$$
2. Dado $a \in \mathbb{Z}$, definimos
$$P : Z_d \to Z_d : m \mapsto P(m) = (m \cdot a)(mod\ d)\,.$$

Ambas não servem a nosso fim, nem para d muito grande.

[5] Dados recentes dizem que o tamanho de toda a nuvem computacional é de 10^{18} bites, logo conteria apenas uma pequena parte de 10^{100} números (grandes!)

1. Neste caso P é bijetora, mas sua inversa se calcula imediatamente visto que a deve ser conhecido por quem conhece P e o oposto de a é $d-a$:

$$S : Z_d \to Z_d : S(c) = (c-a)(mod\ d)\,.$$

   Observe que o código de César descrito em (4.1) é um exemplo deste tipo com $d = 24$ e $a = 3$.

2. Neste caso podemos garantir que P seja bijetora por exemplo se d é primo. Porém, a dificuldade em calcular S conhecendo P é apenas a de calcular o inverso a^{-1} de a, que come vimos pode ser feito facilmente até com d muito grande via algoritmo de Euclides estendido:

$$S : Z_d \to Z_d : S(c) = (c \cdot a^{-1})(mod\ d)\,.$$

Veremos que o que sim funcionará será a operação de **potência a módulo** d: algo da forma

$$P : Z_d \to Z_d : m \mapsto P(m) = (m^e)(mod\ d)\,,$$

com e e d oportunos (e d da ordem de 10^{300}). ★

4.3 Pre-codificação

As funções $P, S : Z_d \mapsto Z_d$ que procuramos apenas transformarão um número em outro. Para codificar mensagens precisaremos antes fixar uma forma de transformar a mensagem num número e vice-versa.

Esta operação, dita **pre-codificação**, não devera conter nenhum secreto: será um algoritmo (público), prefixado e conhecido por todos.

Mais precisamente, daqui para frente assumiremos como dado um código que, dado $d \in \mathbb{N}$ da ordem de 10^{300}, permita transformar uma mensagem em um (ou mais) número $m \in Z_d$, e vice-versa.

O problema da criptografia torna-se então o de determinar as funções $P, S : Z_d \to Z_d$ com as propriedades descritas anteriormente.

Exemplo 4.4. Por exemplo, uma forma de pre-codificar seria escrever todos os códigos ascii da mensagem e interpretar como um único número, que será então o m.

Ao receber e decriptar a mensagem, o destinatário deverá apenas pegar os dígitos a 3 a 3 e converter de volta em letras.

Se a mensagem for comprida demais deverá ser quebrada (pensando d de 300 dígitos seria possível enviar mensagem de até 100 caracteres em um único $m \in Z_d$. ★

4.3.1 Padding

Normalmente o algoritmo de pre-codificação inclui também o chamado **padding**. Isso consiste em adicionar à mensagem, de forma padronizada, um preenchimento.

Este preenchimento costuma ter uma parte fixa e uma parte aleatória, por exemplo, querendo enviar a mensagem "SOS", o padding a transformará na mensagem "mensagem SOS abcdef", sendo abcdef caracteres gerados casualmente.

Veremos na Seção 4.7 vários motivos para fazer o padding, mas um deles já foi mencionado quando descrevemos o envio de mensagens assinadas, na Seção 4.2: se com o padding descrito acima, ao calcular $m = P(a)$, a mensagem obtida não começa com "mensagem", o recebente saberá imediatamente que algo deu errado e então a não foi mesmo produzido como $S(m)$. O mesmo acontece se tiver algum problema no processo (como erros de transmissão ou até mesmo falhas no algoritmo de criptografia).

4.4 O RSA

O RSA é um método de criptografia de chave pública, inventado por **Ron Rivest, Adi Shamir, Leonard Adleman (77)**.

Para apresentar o método RSA vamos introduzir os seguintes ingredientes:

- p, q dois primos grandes (da ordem de 10^{150}) (**secretos**),
- $n = pq$ (**público**), $\phi = (p-1)(q-1)$ (**secreto**),
- $e \in Z_\phi \setminus \{1\}$ invertível ($MCD(e, \phi) = 1$) (**público**) ,
- $f = e^{-1}$ em Z_ϕ (**secreto**).

É importante fazer algumas observações.

- Sabendo ϕ e n não seria difícil calcular p e q. De fato $\phi = (p-1)(q-1) = pq-p-q+1 = n+1-(p+q)$, logo deduziríamos o valor da soma $p+q$, ma o problema de calcular dois números conhecendo sua soma e seu produto se resolve apenas com uma equação de segundo grau: mesmo com incógnitas de 150 dígitos é um problema facilmente resolúvel com um computador. De fato $(x-q)(x-p) = x^2-(p+q)+pq$ e logo
$$p, q = \frac{p+q \pm \sqrt{(p+q)^2 - 4pq}}{2} .$$
- Sabendo p, q calcula-se ϕ, com isso para achar um e invertível e seu inverso usa-se o Algoritmo de Euclides estendido (computacionalmente fácil).

- sabendo só (n, e) descobrir um qualquer entre p, q, ϕ, f equivale a fatorar n (computacionalmente difícil)[6] [7].

Com os ingredientes acima, as funções P e S do método RSA são definidas no conjunto Z_n da seguinte forma:

- **CODIFICAÇÃO**: usando a **chave pública** (n, e):
  dada a mensagem $m \in Z_n$, a codificação é $c = m^e \ (mod\ n)$, isto é:

$$P : Z_n \to Z_n : m \mapsto m^e \ (mod\ n)\,.$$

- **DECODIFICAÇÃO**: usando a **chave secreta** (n, f):
  dada a mensagem codificada $c \in Z_n$ a decodificação é $d = c^f \ (mod\ n)$, isto é:

$$S : Z_n \to Z_n : c \mapsto c^f \ (mod\ n)\,.$$

Resumindo, para construir um método de encriptação RSA precisa gerar dois primos grandes p, q. Feito isso precisará calcular n, ϕ, escolher e e calcular f (todas operações viáveis para quem conhece p, q). Enfim, a chave pública (n, e) será divulgada enquanto os outros dados serão mantidos secretos. [8]

Observação 4.5. Os requisitos mencionados na seção 4.2.1 estão satisfeitos. O requisito ((A)) vale já que P, S são definidas no conjunto Z_n que tem cardinalidade do tamanho de 10^{300}.

Com respeito ao requisito ((B)), veja que, apesar da fórmula para S e para P ser simples de escrever e de implementar num computador (aliás, o mesmo algoritmo que calcula P serve para calcular S, apena trocando os exponentes e, f), seu calculo efetivo precisa do conhecimento do exponente, e como vimos sabendo (n, e) apenas não tem como descobrir f, ou seja, mesmo sabendo aplicar P não tem como deduzir a fórmula para aplicar S. ◁

O funcionamento do método é consequência do teorema abaixo:

Teorema 4.6 (RSA)**.**
$S = P^{-1}$

[6] Descobrir p, q de n significa fatorar, mas vimos que descobrir ϕ leva facilmente a p, q, logo se fosse fácil de achar traria um método de fatorização. O mesmo vale se fosse fácil de achar f, pois isso nos daria facilmente ϕ. Para mais detalhes veja [Cou00, p. 168].

[7] Note que na verdade não temos uma demonstração de que não existam métodos viáveis para quebrar o segredo, apenas por enquanto não foram encontrados.

[8] Note que para a aplicação da função S nem precisa conhecer p, q, ϕ, mas apenas o n (que é público) e o f.

Demonstração. Mostremos para começar que, com as definições dadas,

$$c^f M_p = mM_p \qquad \text{e} \qquad c^f M_q = mM_q. \tag{4.2}$$

Vamos fazer a prova com o módulo p, já que o caso q é análogo.

Usando as propriedades do Lema 3.6 e lembrando que $c = m^e M_n$, calculamos

$$c^f M_p = (m^e M_n)^f M_p = ((m^e M_n) M_p)^f M_p \,.$$

A operação $(xM_n)M_p$ é difícil para genéricos n, p, mas como n é múltiplo di p temos que $(xM_n)M_p = xM_p$. [9] Logo

$$c^f M_p = ((m^e)M_p)^f M_p = (m^e)^f M_p = m^{ef} M_p \,,$$

mas $(ef)M_\phi = 1$, logo $ef = i\phi + 1$ para um oportuno $i \in \mathbb{N}$.

Temos então que

$$c^f M_p = m^{i\phi+1} M_p = ((m^\phi)^i * m) M_p \,.$$

Agora $m^\phi M_p = (m^{p-1} M_p)^{q-1} M_p$ e temos

$$m^{(p-1)} M_p = \begin{cases} 0 & \text{se } p \text{ divide } m, \\ 1 & \text{se } p \text{ não divide } m, \end{cases}$$

de fato, quando p não divide m podemos usar o Pequeno Teorema de Fermat 3.39, enquanto no caso em que p divide m (e o teorema não pode ser aplicado), temos simplesmente que $mM_p = 0$. Em ambos os casos podemos concluir que

$$c^f M_p = mM_p \,.$$

Aplicado (4.2) para q e para p, temos que, pondo $x = c^f - m$,

$$\begin{cases} x \ (mod\ p) = 0, \\ x \ (mod\ q) = 0. \end{cases}$$

Como p, q são relativamente primos (ambos são primos e são distintos), pelo Teorema Chines do resto 3.43, existe um único $\widetilde{x} \in Z_{pq}$ que satisfaz o sistema, e obviamente é $\widetilde{x} = 0$. Isso quer dizer que $x = c^f - m$ é múltiplo de $n = pq$, ou seja, como $m \in \mathbb{Z}_n$, vale $c^f M_n = m$, o que prova que $S \circ P = Id_{Z_n}$.

Para mostrar que $S = P^{-1}$ falta mostrar que também $P \circ S = Id_{Z_n}$, mas isso é demonstrado pelo mesmo raciocínio, já que as fórmulas de S e de P são definidas simetricamente: uma toma a potência com exponente o inverso (em Z_ϕ) da outra, e ambas tomam a mesma operação de módulo. □

[9] De fato, se $x = in + (xM_n) = ipq + (xM_n)$ então $xM_p = (xM_n)M_p$.

Observação 4.7. Na Seção 4.2 vimos como usar dois métodos de criptografia para trocar mensagens que sejam no mesmo tempo criptografadas e assinadas.

Na verdade, em vista da definição do RSA surge um problema: em geral não será possível aplicar $P_A \circ S_B$, já que o domínio $\mathbb{Z}_{n_A}$ das funções de A e o domínio $\mathbb{Z}_{n_B}$ das de B serão necessariamente diferentes (se $n_A = n_B$, pela unicidade da fatoração, significaria que os dois métodos teriam também os mesmos p, q secretos).

Vimos porém que a ideia funcionaria tanto aplicando $S_A \circ P_B$ quanto $P_B \circ S_A$. Precisará então escolher a formulação que aplica primeiro a função cujo n é menor, assim ao resultado poderá sempre ser aplicada a (restrição da) função seguinte. Implementar isso não seria um problema já que n_A e n_B são públicos. ◁

4.5 Algoritmos de cálculo

Vejamos algumas questões técnicas sobre a implementação do RSA.

Tipicamente um computador armazena e faz operações "naturalmente" com variáveis de um certo tamanho. Para variáveis deste tamanho o tempo computacional para realizar uma operação independe do número. Por exemplo, assumindo variáveis de 8 bits, a operação de fazer o produto $a \cdot b$ consiste em enviar a e b para o processador que faz a operação bit por bit: o tempo será igual para multiplicar $3 = (00000011)_2$ por $4 = (00000100)_2$ ou para multiplicar $(10110011)_2$ por $(01110100)_2$.

Ao trabalhar com varáveis com centenas de bits, será necessário desenvolver algoritmos ou usar livrarias que armazenem os números em conjuntos de variáveis naturais e façam as operações com eles.

É claro que o espaço necessário para armazenar cada número será proporcional ao seu tamanho em bits T. O número de operações "naturais" necessárias para fazer a soma de dois tais números será também proporcional a T, mas o número de operações "naturais" necessárias para fazer o produto será da ordem de T^2. Simplificando um pouco, sejam $a_0, .., a_{T-1}, b_0, .., b_{T-1}$ de forma que

$$a = \sum_{i=0}^{T-1} a_i 2^i, \qquad b = \sum_{i=0}^{T-1} b_i 2^i \,,$$

então

$$a + b = \left(\sum_{i=0}^{T-1} a_i 2^i\right) + \left(\sum_{j=0}^{T-1} b_j 2^j\right) = \sum_{i=0}^{T-1} (a_i + b_i) 2^i = ...$$

$$ab = \left(\sum_{i=0}^{T-1} a_i 2^i\right) \left(\sum_{j=0}^{T-1} b_j 2^j\right) = \sum_{i=0}^{T-1} \sum_{j=0}^{T-1} a_i b_j 2^{(i+j)} = ... :$$

para a soma precisamos da T operações $a_i + b_i : \ i = 0, ..T-1$, enquanto para o produto precisamos das T^2 operações $a_i * b_j : \ i, j = 0, ..T-1$.

Além disso, o resultado da soma ocupará um espaço parecido ao ocupado por cada adendo, enquanto o produto ocupará, em geral, um espaço par à soma do espaço ocupado por cada um dos fatores.

4.5.1 Calculo de potência a módulo

Como vimos, o algoritmo que implementa o RSA precisa calcular uma potencia em $\mathbb{Z}_d$, onde tipicamente, base, expoente e d são números de aproximadamente 300 dígitos decimais ($\simeq 1000$ bits).

Obviamente é inviável calcular m^e e depois fazer o módulo, pois m^e poderia ter 10^{300} DÍGITOS ou mais!! Precisamos então tomar o módulo depois de cada operação de produto, desta forma nunca precisaremos calcular nem armazenar números maiores que d^2.

Mesmo assim, multiplicar números entre sí 10^{300} vezes não seria viável. Vejamos então uma forma otimizada de calcular $a^b \ (mod\ d)$.

$a^b \ (mod\ d)$

Suponha $a, b < c < 2^T$ (tipicamente $T \simeq 1000$). Seja $t \leq T$ e

$$b = \sum_{i=0}^{t-1} b_i\, 2^i = \sum_{\substack{i=0,..,t-1 \\ b_i=1}} 2^i,$$

isto é, b_i são os dígitos binários de b.

(1) Ponha $a_0 = a$, $r = a^{b_0}$;

(2) para $i = 1, .., t-1$

  (A) calcule $a_i = a_{i-1}^2 \ (mod\ d)$, desta forma $a_i = a^{2^i} \ (mod\ d)$;

  (B) se $b_i = 1$ ponha $r = r * a_i \ (mod\ d)$.

Note que ao fazer cada produto obtemos um número de tamanho 2^{2T} (precisa cuidar de não ter overflow aqui!) mas depois de tomar o módulo dele sempre nos reduzimos a números de tamanho 2^T.

No final do ciclo teremos

$$r = \left(\prod_{\substack{i=0,..,t-1 \\ b_i=1}} a_i \right) \ (mod\ d) = \prod_{i=0}^{t-1} a^{b_i 2^i} \ (mod\ d) = \left(a^{\sum_{i=0}^{t-1} b_i 2^i} \right) \ (mod\ d) = a^b \ (mod\ d)\,.$$

Cada passo (A) e cada passo (B) envolve um produto de dois números "grandes" e uma operação de módulo. O passo (A) será feito t vezes. O passo (B) será feito tantas vezes quantos são os dígitos 1 de b.

Resumindo, o número de operações (digamos de produtos entre números de tamanho 2^T) que precisamos fazer durante o processo é da ordem de t. Note que o peso computacional é menor se o exponente b for pequeno, mas também se poucos dígitos binários dele são 1, pois teremos menos produtos no passo (B). Por este motivo é frequente escolher como exponente e da chave pública um número pequeno e com poucos dígitos 1, como por exemplo $3 = (11)_2$, $17 = (10001)_2$ ou $65 = (1000001)_2$.

Mesmo assim, o f será calculado como e^{-1} e será em geral do tamanho de d e com muitos dígitos 1. Porém em geral será apenas o computador da entidade dona do secreto que deverá fazer a conta mais pesada com f, deixando a carga dos computadores dos clientes mais baixas.

Para comparar o peso computacional entre métodos com diferentes tamanhos de chave, lembre que vimos que os produtos entre números de tamanho 2^T tem um peso $\simeq T^2$, logo o peso do algoritmo será $\simeq T^3$, que pode reduzir-se a $\simeq tT^2$ escolhendo um exponentes com apenas t bits como comentado acima.

4.6 Procurando primos*

Vimos nas seções anteriores que para produzir um bom método RSA é necessário encontrar números primos de centenas de dígitos[10].

Já vimos que os primos são infinitos, na verdade, eles não são raros:

Teorema 4.8 (Teorema dos números primos, 1896)**.**
Seja

$$\pi(N) = \textit{o número de primos menores que } N.$$

Então

$$\lim_{N\to\infty} \frac{\left(\frac{\pi(N)}{N}\right)}{\left(\frac{1}{\ln(N)}\right)} = 1\,.$$

O Teorema apenas nos dá um comportamento assintótico, mas significa que, para N grande, tem em média um número primo a cada $\ln(N)$ números, ou seja, escolhendo un N grande ao acaso, **a probabilidade de N ser primo é cerca de $\frac{1}{\ln(N)}$**.

Isso significa que se estamos procurando por um primo de 100 dígitos, podemos pegar $\ln(10^{100}) \simeq 230$ números consecutivos de 100 dígitos e muito provavelmente um deles será primo. Pegando 1000 estaremos quase certos de ter um primo entre eles.

O problema porém é que não será fácil reconhecer este primo: vimos que tentar fatorá-lo (se ele for primo mesmo) é computacionalmente inviável. Por isso **precisamos um teste rápido para verificar se um número é primo**.

Exemplo 4.9. Escrevendo um pequeno programa (para números deste tamanho a fatoração é viável), podemos ver que os primeiros primos depois de 10^9 são 10^9+7, 10^9+9 , 10^9+21, 10^9+33 e 10^9+87: podemos ver que a distribuição não tem nenhuma regularidade, mas de fato encontramos 5 primos no espaço de 100 unidade, o que é comparável com o valor de $5 * \ln(10^9) \simeq 100$ que a estimativa do teorema fornece. ★

[10] Os assuntos desta seção podem ser encontrados em [Cou00]

4.6.1 Testes de primalidade, pseudoprimos

A contrapositiva do Pequeno Teorema de Fermat 3.39, diz o seguinte

> Se existe $w \in \mathbb{Z}_d \setminus \{0\}$ tal que $w^{d-1} \ (mod\ d) \neq 1$ então d é composto.

Esta afirmação forneceria uma forma de garantir que um número d não é primo: seria suficiente achar um w com a propriedade acima.

Em vista disso fixamos as seguintes **definições**.

- Chamamos **testemunha**, um $w \in \mathbb{Z}_d \setminus \{0\}$ tal que $w^{d-1} \ (mod\ d) \neq 1$.
  Com isso queremos dizer que w é testemunha o fato que d deve ser composto.
- Se d é composto, $1 < b < d-1$ e $b^{d-1} \ (mod\ d) = 1$, dizemos que **d é pseudoprimo com respeito à base** b.
  Com isso queremos dizer que, apesar de não ser primo, nesta base tem o comportamento que todos os primos tem pelo PTdF[11].

> Para provar que d é composto, em vez de procurar por divisores de d podemos procurar por testemunhas.

De fato, o Lema a seguir mostra que se d não é primo certamente existem testemunhas.

Lema 4.10. *Se $d \in \mathbb{N} \setminus \{1\}$ não é primo e $\gamma \in \mathbb{Z}_d$ não possui inverso, então*

$$\gamma^{d-1} \neq [1]_d \,.$$

Demonstração. Por contradição, se valesse $\gamma^{d-1} = [1]_d$ então $\gamma^{d-2} \cdot \gamma = [1]_d$ e logo γ^{d-2} seria o inverso de γ. □

Observação 4.11. A priori não dá para dizer se este método é bom: de fato a existência de pseudoprimos diz que às vezes pode ser difícil encontrar testemunhas mesmo se d for composto.

Na verdade, se os únicos testemunhas fossem os não invertíveis, não teria nenhuma vantagem neste método, já que, que como sabemos, um não invertível é um número que tem fatores em comum com d, logo encontrá-lo seria tão difícil quanto fatorar d.

Fortunadamente a condição do Lema 4.10 é apenas suficiente, e em geral existem muitos mais testemunhas que divisores.

Um test sobre números pequenos mostra que o método é promissor, de fato se calculamos $2^{d-1} \ (mod\ d)$ para todos os d de 2 a 100, descobrimos que o resultado é 1 se e só se o número é primo.

Infelizmente isso não vale sempre: veja o exemplo a seguir. ◁

[11] Note que excluímos desta definição 1 e $d-1$ já que ambos sempre dão 1 quando elevados a $d-1$ (se d é ímpar).

Exemplo 4.12.

- $2^{340} \ (mod\ 341) = 1$, sendo $341 = 11 \cdot 31$.
  Isso significa que 341 é composto, mas pseudoprimo com respeito à base 2.
  Porém $3^{340} \ (mod\ 341) = 56$, logo só testando mais uma base achamos uma testemunha para 341.

- $561 = 3 \cdot 11 \cdot 17$ é pseudoprimo com respeito a toda base prima com ele (as não primas já sabemos que são necessariamente testemunhas pelo Lema 4.10). Ou seja, as únicas testemunhas são os não invertíveis. Números assim são ditos **de Carmichael**, e existem infinitos deles (561 é o menor).

★

Em resumo, o método parece bom (é infalível entre 2 e 99), mas também não é perfeito, devido à existência dos números de Carmichael.

Podemos ainda testar números um pouco maiores, para os quais ainda é possível obter a fatoração computacionalmente com o algoritmo da Seção 3.4.1. Descobrimos que dos números entre 2 e 10^9:

- 50'847'534 são primos,
- 5597 são pseudoprimos com respeito à base 2,
- 1272 são pseudoprimos com respeito às bases 2 e 3,
- 685 são pseudoprimos com respeito às bases 2, 3 e 5,
- 646 são números de Carmichael,
- *1282 são pseudoprimos fortes com respeito à base 2 (veja a definição na seção 4.6.2).*

Isso significa que se $d < 10^9$ e $2^{d-1} \ (mod\ d) = 1$ a probabilidade que d seja primo é $99,99\%$ (apenas 5597 são compostos dos mais de $50 \cdot 10^6$ que satisfazem $2^{d-1} \ (mod\ d) = 1$). Se ainda tivermos que $3^{d-1} \ (mod\ d) = 1$, a probabilidade seria maior ainda, mas infelizmente só poderemos ter a certeza absoluta de d ser primo se não encontrarmos uma testemunha menor que $\sqrt{d}$ (em vista do Lema 4.10 e da existência dos números de Carmichael).

Apesar dos aparentemente bons resultados descritos acima, este método **não é satisfatório**, pelo fato que não temos nenhuma base teórica que nos diga se o método será tão bom para números de 150 dígitos como vimos ser para números de 9 dígitos, em particular não poderemos estimar a probabilidade de um número ser primo já que também não seria viável fazer uma estatística como a acima para TODOS os números deste tamanho.

Por este motivo veremos na próxima seção uma modificação deste método, que permitirá sim de estimar a probabilidade do número ser primo, graça a um resultado teórico (veja o Teorema 4.15).

4.6.2 Teste de Miller-Rabin:

Vejamos nesta seção o chamado Teste de Miller-Rabin. O teste se baseia na seguinte propriedade:

Observação 4.13. Seja $d > 2$ um número primo.

- Sempre podemos escrever $d = 2^k q + 1$ com q ímpar, $k \geq 1$.
- Dada uma base $b \in Z_d \setminus \{0\}$, pelo PTdF $b^{2^k q} \ (mod\ d) = 1$, logo a sequência

$$b^{2^i q} \ (mod\ d) = 1 : \quad i = 0, .., k \tag{4.3}$$

  termina por 1.
- Também vale um dos seguintes:
  - a sequência (4.3) começa por 1 e logo vale sempre 1.
  - a sequência (4.3) vale $d - 1$ antes do primeiro 1.

Prova da alternativa. Como $b^{2^{i+1} q} = (b^{2^i q})^2$, é claro que cada elemento da sequência é o quadrado (em $\mathbb{Z}_d$) do anterior. Por isso, depois do primeiro 1 todos os elementos seguintes serão também 1.

Suponhamos que x seja o último elemento da sequência antes do primeiro 1, então (em $\mathbb{Z}_d$) $x^2 = 1$, ou seja $x^2 - 1 = (x+1)(x-1) = 0$ (verifique que as passagens algébricas podem ser feitas em $\mathbb{Z}_d$, em vista do Lema 3.6).

Como d é primo, $\mathbb{Z}_d$ é um corpo e logo vale a regra do anulamento do produto, isto é, $x = \pm 1$, mas assumimos $x \neq 1$ e logo $x = -1 = d - 1$ (tudo em sentido $\mathbb{Z}_d$). □

Observe que se d não é primo, a última passagem da prova pode ser substituída pela afirmação que um entre $x \pm 1$ é um elemento não invertível.

O **Teste de Miller-Rabin** (TMR) consiste em verificar a propriedade descrita: se o comportamento não for o descrito então o número não é primo.

Em analogia com a seção anterior definimos:

- diremos que **o número d satisfaz o TMR para uma certa base b** se a sequência (4.3) segue a regra descrita na Observação 4.13;
- se d é composto, $1 < b < d - 1$ e a sequência $b^{2^i q} \ (mod\ d)$ satisfaz (TMR) dizemos que **d é fortemente pseudoprimo com respeito à base b**.

Temos então o seguinte:

- se (TMR) vale para a base b, então d é primo ou fortemente pseudoprimo com respeito à base b,
- se (TMR) não vale então d é composto.

Exemplo 4.14. • Seja $d = 341$. Como $d - 1 = 340 = 2^2 * 85$ temos que a sequência a considerar será b^{85}, b^{2*85} e $b^{3 \cdot 85}$.

Podemos facilmente escrever um programa que executa o test. Para 341 temos

```
n=341=85*2^2+1
2^85 (mod 341)=32
2^170 (mod 341)=1
2^340 (mod 341)=1
```

O teste não está satisfeito, já que $32 \neq 340$. Concluímos que 341 não é primo nem fortemente pseudoprimo na base 2 (apesar de ser pseudoprimo na base 2).

Veja que o 32 não é um número qualquer, de fato em $\mathbb{Z}_{341}$, $32^2 = 1$, ou seja, $31 * 33 = 0$ (ambos são não invertíveis, já que $341 = 11 * 31$!).

- Teste de 25 na base 7:

```
n=25=3*2^3+1
7^3 (mod 25)=18
7^6 (mod 25)=24
7^12 (mod 25)=1
7^24 (mod 25)=1
```

Concluímos que o composto 25 é fortemente pseudoprimo na base 7.

- Teste de 37 na base 2:

```
n=37=9*2^2+1
2^9 (mod 37)=31
2^18 (mod 37)=36
2^36 (mod 37)=1
```

O teste está satisfeito, já que 37 é primo.

- Teste de 31 na base 2:

```
n=31=15*2^1+1
2^15 (mod 31)=1
2^30 (mod 31)=1
```

O teste está satisfeito, já que 31 é primo; neste caso a sequência vale 1 desde o primeiro termo.

- Teste de 39 na base 1:

```
 n=39=19*2^1+1
2^19  (mod  39)=11
2^38  (mod  39)=4
```

  Neste caso o teste não está satisfeito, de fato 39 é composto.

★

Como visto no exemplo, pseudoprimos fortes existem (por exemplo, entre 1 e 10^9 temos 1282 pseudoprimos fortes com respeito à base 2, o primeiro é 2047), por esta razão se d não satisfaz o TMR podemos concluir com certeza que d é composto, ma se satisfaz o TMR não podemos ainda ter certeza que seja primo (como com o teste da seção anterior).

A verdadeira superioridade do Test de Miller-Rabin está no fato que existe o seguinte resultado teórico, que vale para números de qualquer tamanho.

Teorema 4.15 (Teorema de Rabin, 1980)**.** *Seja d ímpar. Se (TMR) vale para mais que $d/4$ bases entre 1 e $d-1$, então d é primo.*

O Teorema ainda não permite chegar à certeza que d seja primo (pois testar $d/4+1$ bases ainda é inviável), mas permite uma abordagem probabilística.

Dado um d qualquer e uma base b temos três possibilidades:

1. d é primo e logo TMR vale na base b,
2. d é composto mas TMR vale na base b,
3. d é composto e TMR não vale na base b.

Se acontecer o evento 3 concluímos que d é composto, em caso contrário não sabemos se estamos no caso 1 ou 2.

Mas do teorema de Rabin sabemos que se d é composto, (TMR) pode valer para, no máximo, um quarto das bases. Dito de outra forma, dado um número composto a probabilidade que (TMR) valha numa certa base b é no máximo $1/4$, ou seja $4P(2) < P(2)+P(3)$, além disso $P(2)+P(3) = 1-P(1) < 1$, do que concluímos que $P(2) < 1/4$.

Repetindo o teste em outra base, de novo a probabilidade de acontecer o item 2 sera menos que $1/4$, mas se gerarmos as bases aleatoriamente podemos assumir que os resultados sejam independentes, logo a probabilidade de cair no caso 2 duas vezes será $< 1/16$, e generalizando, repetindo com k bases aleatórias a probabilidade de cair no caso 2 todas as vezes será $< 1/4^k$.

Como o caso 1 é o que desejamos e quando cairmos no caso 3 sabemos com certeza que d é composto, podemos concluir que se desejamos manter a probabilidade do evento indesejável "d é composto mas passou no teste todas as vezes" abaixo de um nível $\varepsilon > 0$,

deveremos repetir o teste pelo menos k vezes com $k > -\log_4 \varepsilon$. Por exemplo, para uma probabilidade menor que 1 sobre um milhão serão suficientes 10 bases.

Em conclusão, aplicando o TMR um número suficiente de vezes, **podemos chegar a uma probabilidade alta quanto quisermos que d seja primo.**

4.6.3 Rotina para escolha dos primos para RSA

Juntando quanto visto até agora, vejamos a típica rotina para escolher os primos de um método RSA:

- escolha o numero de dígitos para n, divida eles entre p e q de forma não igual[12];
- escolha N aleatoriamente[13] do tamanho desejado para p e considere cerca de $3\ln(N)$ números a seguir;
- procure divisores pequenos para eliminar rapidamente a maioria destes números;
- use o Teste de Miller-Rabin nos que sobram com um número de bases que deixe a probabilidade de erro pequena quanto queira;
- se nenhum número passa o teste tente de novo com outro N (improvável);
- uma vez encontrado o candidato ainda faça tentativas de fatorá-lo[14] ...para maior segurança;
- repita para q.

4.7 Algumas possíveis fraquezas e soluções.

4.7.1 Mensagem pequena

Vimos que frequentemente e é escolhido pequeno para diminuir o peso computacional da codificação.

Isso porém traz um perigo, pois se $m^e < n$ então a mensagem encriptada será exatamente $c = m^e$ (no sentido de $\mathbb{R}$) e o cálculo da raiz e-ésima em $\mathbb{R}$ é uma operação simples: a decodificação será fácil!

Uma solução para isso é usar o padding (veja Seção 4.3.1) para deixar a mensagem comprida.

Por exemplo, se enviarmos a mensagem "ok" com a pre-codifica descrita no exemplo 4.4, teremos uma mensagem de apenas 6 dígitos: com $e = 17$ a codificação $c = m^e$

[12] Como existem técnicas de fatoração eficientes para achar eventuais fatores perto da raiz do número, é melhor escolher p, q de tamanhos diferentes, por exemplo com 140 e 160 dígitos, respectivamente.

[13] Se escolhermos o primeiro primo depois de 10^{150} seria fácil de adivinhar.

[14] é importante atacar o número com todos os algoritmos conhecidos que podem fornecer uma fatoração em casos particulares.

terá aproximadamente 100 dígitos e será menor que d em um método de 300 dígitos, resultando facilmente decifrável. Mas se enviarmos "mensagem ok abcdef", já daria um m de 50 dígitos e m^e amplamente acima de 300 dígitos, obtendo uma codificação mais segura.

4.7.2 Mensagem padrão

Imagine que se queira interceptar uma mensagem já sabendo que ela só poderá ser "sim" ou "não". Como a função de codificação é pública, o interceptador precisará apenas produzir a codificação das duas possíveis respostas e comparar com a interceptada.

Uma solução para isso é o fato de inserir caracteres aleatórios no padding. Desta forma as mensagens "sim" e "não" poderão ser pre-codificadas em uma grande quantidade de diferentes mensagens m, deixando impossível comparar com todas as possíveis $P(m)$.

4.7.3 p não primo

Como vimos, os métodos para produzir primos nunca nos darão certeza absoluta que os p, q escolhidos sejam realmente primos.

Se acontecer de um deles não ser primo, obviamente a fatoração de n será mais fácil (mas também não tão fácil assim pois o número passou todas as tentativas de fatorá-lo feitas na sua produção).

Apesar disso, o método ainda funciona bastante bem: de fato, lembre que a prova do funcionamento do método se baseava no PTdF, que garanta que $m^{p-1}M_p = 1$, mas mesmo se p não for primo, deve ser um primo com poucas testemunhas já que passou pelo TMR, logo $m^{p-1}M_p = 1$ ainda para muitos m. Quanto ao Teorema Chines do Resto, ainda pode ser aplicado se p e q são primos entre eles (seria bem difícil ter escolhidos falsos primos e ainda com fatores comuns!!)

Caso o método não funcione, então teríamos $S(P(m)) \neq m$, dando uma mensagem sem significado, o que indicaria que um dos fatore não é primo e que precisamos mudar de sistema, mas sem que isso tenha posto em perigo a comunicação no sentido de deixar que mensagens secretas fossem legíveis (apenas poderiam tornar-se totalmente ilegíveis).

4.7.4 Mensagens com múltiplos destinatários e e pequenos

Suponha que seja preciso enviar a mesma mensagem m a diferentes destinatários, cada um com seu método RSA, e que os 3 métodos tenham $e = 3$ (o que seria possível visto que e em geral é escolhido simples, e fácil de verificar já que e faz parte da chave pública).

Sejam n_1, n_2, n_3 o n das três chaves públicas. Eles serão quase certamente primos entre eles (lembre que a geração dos primos começa com um número casual!)

Se alguém interceptar as três mensagens, então terá

$$\begin{cases} m^3 \ (mod \ n_1) = c_1, \\ m^3 \ (mod \ n_2) = c_2, \\ m^3 \ (mod \ n_3) = c_3. \end{cases}$$

Pelo Teorema Chinês do resto (lembre que vale até para mais de 2 equações, se os módulos são primos entre eles), m^3 fica univocamente determinado a menos de um múltiplo de $n_1n_2n_3$ (e fácil de calcular, lembre na prova do teorema, a formula dada usando o algoritmo de Euclides estendido).

Mas como $m < n_i, \ i = 1, 2, 3$, temos que $m^3 < n_1n_2n_3$, logo o que calculamos é exatamente m^3 em $\mathbb{R}$, e podemos facilmente calcular sua raiz terceira.

Uma solução para este problema é usar exponentes maiores, mas também o padding com caracteres casuais resolveria o problema, produzindo três mensagens m diferentes.

4.7.5 Exploração do produto

Uma propriedade do RSA é que $P(m_1)P(m_2) = P(m_1m_2)$ (em sentido $\mathbb{Z}_n$), e o mesmo vale para S. Isso pode ser explorado para alguns tipos de ataques.

Suponha interceptar $c = P(m)$ e conseguir com algum engano que o dono do secreto decripte cr^e (aqui r é qualquer número escolhido pelo interceptador).

Então o interceptador tem c, r, e $S(cr^e)$, mas (em sentido $\mathbb{Z}_n$) $S(cr^e) = S(c)S(r^e) = S(c)r$ e então ele obtém $m = S(c) = r^{-1}S(cr^e)$.

[14‡] A LIGEIRA RAPOSA MARROM SALTOU SOBRE O CACHORRO CANSADO

Capítulo 5

Recorrências

5.1 Alguns preliminares

5.1.1 Comportamentos assintóticos a $+\infty$

Fixamos aqui algumas definições que serão usadas, que comparam o comportamento a infinito de duas ou mais funções.

Definição 5.1. *Sejam $f, g : D \to \mathbb{R}$ com $D \subseteq \mathbb{R}$ não limitado para cima, e com $f(x) \geq 0$ e $g(x) > 0$ para x grande.*

- *Diremos*

  $$\text{“}\mathbf{f(x) = O(g(x))} \text{ quando } x \to \infty\text{”}$$

  $$\text{se} \qquad \boxed{\exists\, C, H > 0 : \;\; f(x) \leq C g(x) \qquad \forall x > H.}$$

  Podemos ler isso como f é “O grande" de g quando x tende a ∞, ou f é de ordem menor ou igual a g quando x tende a ∞.

- *Diremos*

  $$\text{“}\mathbf{f(x) = \Theta(g(x))} \text{ quando } x \to \infty\text{”}$$

  $$\text{se} \qquad \boxed{\exists\, c, C, H > 0 : \;\; c g(x) \leq f(x) \leq C g(x) \qquad \forall x > H.}$$

  Podemos ler isso como f é theta de g quando x tende a ∞, ou também f é da mesma ordem de g quando x tende a ∞; outra notação usada é “$f(x) \asymp g(x)$ quando $x \to \infty$”.

- *Diremos*

  $$\text{“}\mathbf{f(x) \sim g(x)} \text{ quando } x \to \infty\text{”}$$

  $$\text{se} \qquad \boxed{\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1\,.}$$

  Podemos ler isso como f é assintótico a g quando x tende a ∞.

Observação 5.2. A primeira definição apenas fornece uma estimativa para cima, enquanto a segunda é mais forte, pois fornece uma estimativa para cima e para baixo.

A última é mais forte de todas, pois implica nas anteriores, com constantes satisfazendo $c < 1 < C$.

Exemplo 5.3. Exemplos: (quando $x \to \infty$)

$$x = O(x^2), \qquad |\sin(x)| = O(x), \qquad x \ln x = O(x^2), \qquad x = O(x \ln x),$$

$$\frac{x}{\ln x} = O(x), \qquad \frac{1}{x} = O(1), \qquad |\sin(x)| = O(1), \qquad 3 + \sin(x) = \Theta(1),$$

$$Ax + B = O(x), \qquad Ax + B = \Theta(x) \qquad para\ A > 0,\ B \in \mathbb{R}\,,$$

$$x + B \sim x \ \ (para\ B \in \mathbb{R}), \qquad \sin(1/x) \sim 1/x\,.$$

★

5.1.2 Algo sobre números complexos

Definição e operações

Definimos um **número complexo** z associando ele a uma dupla de reais, usaremos a seguinte notação:

$$z = x + iy, \quad x, y \in \mathbb{R}\,,$$

onde
– x é dita **parte real** de z ($x = \Re(z)$);
– y é dita **parte imaginária** de z ($y = \Im(z)$).

Definimos no conjunto $\mathbb{C}$ dos números complexos as operações:

$$\textbf{soma}: \qquad (x + iy) +_{\mathbb{C}} (X + iY) := (x + X) + i(y + Y)\,,$$

$$\textbf{produto}: \qquad (x + iy) \cdot_{\mathbb{C}} (X + iY) := (xX - yY) + i(Xy + xY)\,.$$

Desta forma pode ser provado que o conjunto dos números complexos $\mathbb{C}$ forma um **corpo**.[1]

Podemos identificar os complexos $(x+i0)$ com os reais,[2] escrevendo apenas $(x+i0) = x$.

Com esta identificação podemos dizer que os complexos $\pm i = (0 \pm i1)$ são raízes do real -1. [3]

[1] Note em particular que $0 + i0$ é elemento neutro da soma, $1 + i0$ é elemento neutro do produto, $-x - iy$ é o oposto de $x + iy$, enquanto $(x + iy)^{-1} = \frac{x}{x^2+y^2} + i\frac{-y}{x^2+y^2}$.

[2] A identificação pode ser feita pois as operações definidas agem sobre estes complexos exatamente como agem em $\mathbb{R}$, Não poderíamos identificar, por exemplo, os complexos $0 + iy$ com os reais, pois $(0 + i1) \cdot (0 + i(-1)) = (1 + i0) \neq (0 + i(-1))$.

[3] De fato, $(0 + i1) \cdot (0 + i1) = (0 + i(-1)) \cdot (0 + i(-1)) = (1 + i0)$.

Representação geométrica

Podemos representar $z = x + iy \in \mathbb{C}$ no plano $\mathbb{R}^2$ com o ponto de coordenadas (x, y), desta forma o eixo x representa os reais.

A soma em $\mathbb{C}$ pode então ser representada graficamente no plano $\mathbb{C} \sim \mathbb{R}^2$ pela regra do paralelogramo.

Chamamos de **conjugado** de $z = x + iy$, o complexo $\overline{z} = x - iy$. Graficamente é o complexo obtido espelhando z com respeito ao eixo horizontal.

Para interpretar o produto em $\mathbb{C}$ é útil representar um complexo na forma polar:

$$z = x + iy = \rho(\cos(\theta) + i\sin(\theta)),$$

onde $\rho = |z| = \sqrt{x^2 + y^2}$ é dito **módulo** de z e

$$\theta = arg(z) = (2k\pi+) \begin{cases} arctg(y/x) & para\ x > 0, \\ arctg(y/x) + \pi & para\ x < 0, \\ \pi/2 & para\ x = 0,\ y > 0, \\ 3\pi/2 & para\ x = 0,\ y < 0, \\ q.q. & para\ x = 0,\ y = 0, \end{cases} \qquad (k \in \mathbb{Z})$$

é dito **argumento** de z.

Usando fórmulas trigonométricas de adição obtemos a seguinte fórmula para o produto de complexos:

$$[\rho(\cos(\theta) + i\sin(\theta))] \cdot [\sigma(\cos(\phi) + i\sin(\phi))] = [\rho\sigma(\cos(\theta + \phi) + i\sin(\theta + \phi))] \qquad (5.1)$$

e para a potência de um complexo:

$$z^n = \rho^n(\cos(n\theta) + i\sin(n\theta)). \qquad (5.2)$$

O produto em $\mathbb{C}$ então pode ser representado graficamente no plano $\mathbb{C} \sim \mathbb{R}^2$ dizendo que o resultado do produto zw é um vetor em $\mathbb{R}^2$ de modulo o produto dos módulos e argumento a soma dos argumentos (multiplicamos os comprimentos e somamos os ângulos).

Da mesma forma, z^n será um vetor em $\mathbb{R}^2$ de modulo $|z|^n$ e argumento $n \cdot arg(z)$.

Exercício 5.4. Use as fórmulas de produto e potência para verificar que $i^2 = (-i)^2 = -1$ e que -1 possui exatamente 3 raízes cúbicas, sendo elas os complexos de módulo 1 e argumentos, respectivamente, $\pm\pi/3$ e π. ★

5.1.3 Algo sobre polinômios

Um **polinômio a coeficientes reais de grau** r pode ser escrito como

$$\mathcal{P}(x) = c_r x^r + c_{r-1}x^{r-1} + ... + c_1 x + c_0$$

onde $c_r, .., c_0 \in \mathbb{R}$.

- q é dita **raiz** do polinômio $\mathcal{P}$ se $\mathcal{P}(q) = 0$.
  Dada uma raiz q, sempre podemos fatorar $\mathcal{P}(x) = (x - q)\mathcal{Q}(x)$, onde $\mathcal{Q}(x)$ é outro polinômio, de grau menor que $\mathcal{P}$.

- Um polinômio a coeficientes reais de grau r pode possuir até r raízes reais (isto é, $q \in \mathbb{R}$ tal que $\mathcal{P}(q) = 0$).

  Um resultado bem mais preciso é o seguinte.

  Teorema 5.5 (Teorema Fundamental da Álgebra)**.** *Um polinômio $\mathcal{P}$ a coeficientes reais (ou complexos) de grau r possui sempre exatamente r raízes complexas (isto é, $q \in \mathbb{C}$ tal que $\mathcal{P}(q) = 0$), contadas com sua multiplicidade.*

  Isto significa que $\mathcal{P}$ pode sempre ser fatorado em r fatores simples na forma

  $$\mathcal{P}(x) = c_r(x - q_1)(x - q_1).....(x - q_r)\,,$$

  onde $q_1, .., q_r \in \mathbb{C}$ são as raízes. A **multiplicidade** de uma raiz q é o número de vezes que o fator $(x - q)$ aparece na fatoração.

- Se o polinômio $\mathcal{P}$ é a coeficientes reais então as raízes não reais aparecem sempre em duplas de raízes complexas conjugadas, na forma

  $$q = a + ib, \quad \overline{q} = a - ib : \qquad a, b \in \mathbb{R}\,.$$

  De fato podemos observar que $(x-q)(x-\overline{q}) = x^2-(q+\overline{q})x+q\overline{q}$ onde os coeficientes $q + \overline{q} = 2a$ e $q\overline{q} = a^2 + b^2$ são reais.

Exemplo 5.6. $x^2 - 1 = (x + 1)(x - 1)$: duas raízes simples reais: 1 e -1;
$x^2 + 1 = (x + i)(x - i)$: duas raízes simples complexas conjugadas: i e $-i$;
$x^2 + 2x + 1 = (x + 1)(x + 1)$: uma raiz real: -1, de multiplicidade 2;
$x^2 - 3x + 2 = (x - 2)(x - 1)$: duas raízes simples reais: $r_{1,2} = \frac{3\pm\sqrt{9-8}}{2} = 1, 2$;
$x^2+x+1 = (x-r_1)(x-r_2)$: duas raízes simples complexas conjugadas: $r_{1,2} = \frac{-1\pm\sqrt{1-4}}{2} = -\frac{1}{2} \pm i\frac{\sqrt{3}}{2}$. ★

5.2 Recorrências

Uma **equação de recorrência de ordem r** é uma relação que determina cada termo a_n de uma sequência, em função de n e dos r termos anteriores:

$$a_n = f(n, a_{n-1}, a_{n-2}, ..., a_{n-r})\,.$$

Resolver uma equação de recorrência, significa encontrar uma fórmula explicita para a sequência.

Para resolver uma equação de recorrência, precisamos sempre de r **condições iniciais**.

5.2.1 Método de iteração

O método mais fácil para tentar resolver uma recorrência é o **Método de iteração**: consiste em calcular sequencialmente os termos de a_n a partir das condições iniciais e deduzir deles uma fórmula explicita, que depois deverá ser provada por indução.

Exemplo 5.7.

$$\begin{cases} a_n = (n-1)a_{n-1}\,, \\ a_1 = 1\,. \end{cases}$$

Podemos calcular $a_2 = 1 \cdot a_1 = 1$, $a_3 = 2 \cdot a_2 = 2$, $a_4 = 3 \cdot a_3 = 6$, $a_5 = 4 \cdot a_4 = 24$, ...

A regra "parece" ser $a_n = (n-1)!$

Para mostrar que isso vale para todo $n \in \mathbb{N}$ usamos indução, de fato:

– caso base: $a_1 = 1 = 0!$

– passo de indução: se $a_n = (n-1)!$ então

$$a_{n+1} = (n+1-1)a_n = n(n-1)! = n!\,,$$

onde a primeira igualdade sai da regra da recorrência e a segunda da hipótese de indução $a_n = (n-1)!\,.$ ★

Exemplo 5.8.

$$\begin{cases} a_n = a_{n-1}^2\,, \\ a_1 = 1 \text{ (resp. } a_1 = 2)\,; \end{cases}$$

– com $a_1 = 1$ obtemos a seq. constante 1

– com $a_1 = 2$ obtemos a seq. $a_1 = 2$, $a_2 = 4$, $a_3 = 16$, $a_4 = 16^2, ...$, que pode ser escrita (verifique) como $a_n = 2^{2^{n-1}}$. ★

Exemplo 5.9.

$$\begin{cases} a_n = \alpha a_{n-1} + \beta\,, \\ a_0 = c\,. \end{cases}$$

Podemos calcular

$a_1 = \alpha c + \beta$,

$a_2 = \alpha a_1 + \beta = \alpha(\alpha c + \beta) + \beta = \alpha^2 c + \alpha\beta + \beta$,

$a_2 = \alpha a_2 + \beta = \alpha(\alpha^2 c + \alpha\beta + \beta) + \beta = \alpha^3 c + \alpha^2\beta + \alpha\beta + \beta$.

A regra "parece" ser

$$a_n = \alpha^n c + \beta \sum_{i=0}^{n-1} \alpha^i = \alpha^n c + \beta \frac{\alpha^n - 1}{\alpha - 1}\,,$$

exceto no caso $\alpha = 1$ onde $a_n = \alpha^n c + n\beta$. De novo isso deverá ser mostrado por indução.

Note que o termo $S_n := \sum_{i=0}^{n-1} \alpha^i$ satisfaz as estimativas a seguir:

$$|S_n| = \begin{cases} O(|\alpha|^n) & se\ |\alpha| > 1\,, \\ O(1) & se\ -1 \leq \alpha < 1\,, \\ O(n) & se\ \alpha = 1\,. \end{cases}$$

★

Exemplo 5.10.

$$\begin{cases} a_n = 2a_{n-1} - a_{n-2}\,, \\ a_1 = 1,\ a_2 = 3\,. \end{cases}$$

Iterando a recorrência obtemos a sequência 1,3,5,7,9,..: podemos verificar por indução que se trata da sequência $a_n = -1 + 2n$. ★

Exemplo 5.11.

$$\begin{cases} a_n = a_{n-1} + a_{n-2}\,, \\ a_1 = 1,\ a_2 = 1\,. \end{cases}$$

Iterando a recorrência obtemos a sequência (de Fibonacci) 1,1,2,3,5,8,13,21,34,55,89,....
A primeira vista parece difícil encontrar uma fórmula explicita para esta sequência, mas veremos mais a frente como obtê-la (veja no último ponto do exemplo 5.18). ★

5.2.2 Motivações

Muitos problemas práticos são naturalmente modelados por uma recorrência

Exemplo 5.12. • Torre de Hanoi:

Consiste em uma base contendo três pinos, em um dos quais são dispostos N discos, um sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar, um a um, todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma passagem.

Qual é o número mínimo de movimentos para passar os N pinos?

Não é fácil responder diretamente, porém é fácil verificar que se a resposta é C_N, então $C_{N+1} = 2C_N + 1$, já que podemos primeiro transferir N discos até outro pino (C_N movimentos), depois transferir o disco maior para o terceiro pino (1 movimento) e enfim transferir os N discos sobre o disco maior (C_N movimentos).
Além disso, $C_0 = 0$ e logo C_N é definido pela recorrência

$$\begin{cases} C_n = 2C_{n-1} + 1\,, \\ C_0 = 0\,, \end{cases}$$

cuja solução é, pelo ex 5.9, $C_n = 2^n \cdot 0 + 2^n - 1 = 2^n - 1$.

- Quantas regiões no plano podem ser obtidas (no máximo) traçando N retas?

  De novo é fácil ver que se já tem N retas desenhadas, ao traçar mais uma reta obteremos uma região adicional, e mais uma por cada reta que cruzaremos, ou seja se a resposta é R_N, então $R_{N+1} = R_N + N + 1$, sendo que $R_0 = 1$.
  Temos então $R_1 = 2$, $R_2 = 2 + 2 = 4$, $R_3 = 4 + 3 = 7$, ou seja,

$$R_n = 1 + \sum_{i=1}^{n} i = 1 + \frac{n(n+1)}{2}\,.$$

- Outra aplicação em que aparecem recorrências é o estudo do peso computacional de programas de tipo "divide e conquista", veja seção 5.4. ★

5.3 Recorrências lineares

Nesta seção estudaremos as recorrências lineares e veremos que em certos casos podemos encontrar todas as suas soluções explicitamente.

Dizemos que uma recorrência é

- **linear** quando f é linear nos termos da sequência:

$$a_n = C_1(n)a_{n-1} + C_2(n)a_{n-2} + ... + C_r(n)a_{n-r} + g(n)\,; \tag{L}$$

- **linear e homogênea** quando em (L) $g(n) = 0$:

$$a_n = C_1(n)a_{n-1} + C_2(n)a_{n-2} + ... + C_r(n)a_{n-r}\,; \tag{LH}$$

- **linear a coeficientes constantes** quando os coeficientes $C_1, ..., C_r$ não dependem de n

$$a_n = C_1 a_{n-1} + C_2 a_{n-2} + ... + C_r a_{n-r} + g(n)\,; \tag{LCC}$$

- **linear a coeficientes constantes e homogênea**: quando

$$a_n = C_1 a_{n-1} + C_2 a_{n-2} + ... + C_r a_{n-r}\,. \tag{LCCH}$$

Exemplo 5.13. Entre as recorrências vistas,

$a_n = 2a_{n-1} - a_{n-2}$ é (LCCH).

$a_n = \alpha a_{n-1} + \beta$ e $R_{N+1} = R_N + N + 1$ são (LCC), a primeira é homogênea se e só se $\beta = 0$.

$a_n = (n-1)a_{n-1}$ é (LH), mas não a coef. const.

$a_n = a_{n-1}^2$ não é linear!! ★

O motivo pelo qual o estudo das equações lineares é mais fácil, nos permitindo chegar em muitos casos a fórmulas completas de resolução, é a seguinte propriedade, chamada **Princípio de sobreposição**:

- Se a_n e b_n satisfazem a mesma recorrência (LH) então
  $Aa_n + Bb_n$ também satisfaz a mesma recorr, $\forall A, B \in \mathbb{R}$;
- se p_n satisfaz uma recorrência (L) e a_n satisfaz a recorrência **homogênea associada** (ou seja, a (LH) obtida da (L) retirando o termo $g(n)$), então
  $a_n + p_n$ também satisfaz a mesma recorr. (L);
- se p_n satisfaz uma recorrência (L) com $g = g_1$ e q_n satisfaz a mesma recorrência (L) mas com $g = g_2$, então
  $p_n + q_n$ satisfaz a mesma recorrência (L), mas com $g = g_1 + g_2$;
- se p_n satisfaz uma recorrência (L) com certa g, então
  Dp_n satisfaz a mesma recorrência (L) mas com Dg no lugar de g, $\forall D \in \mathbb{R}$.
- O mesmo vale com (LCC) e (LCCH).

Exemplo 5.14. Verifique que $2^n - 1$ e $3 \cdot 2^n - 1$ são ambas soluções de $a_n = 2a_{n-1} + 1$, enquanto a soma delas satisfaz $a_n = 2a_{n-1} + 2$.

Verifique que $a_n = 1$ e $a_n = n$ são ambas soluções de $a_n = 2a_{n-1} - a_{n-2}$, e também são soluções todas as comb.lin. $a_n = A + Bn$ com $A, B \in \mathbb{R}$.

Verifique que $a_n = 2^{2^{n-1}}$ e $a_n = 1$ são ambas soluções de $a_n = a_{n-1}^2$, mas a soma delas não é. ★

5.3.1 Solução de recorrências lineares homogêneas a coeficientes constantes

Vamos procurar uma fórmula de resolução para as recorrências lineares homogêneas a coeficientes constantes na forma (LCCH).

Para obter uma solução[4], por analogia com o exemplo 5.9 quando $\beta = 0$, procuramos soluções na forma $a_n = \lambda^n$ com λ para determinar.

Substituindo em (LCCH) temos

$$\lambda^n = C_1\lambda^{n-1} + C_2\lambda^{n-2} + ... + C_r\lambda^{n-r}\,,$$

eliminando o fator comum λ^{n-r} chegamos à **equação característica** associada à recorrência (LCCH):

$$\lambda^r = C_1\lambda^{r-1} + C_2\lambda^{r-2} + ... + C_r\lambda^0\,. \qquad \text{(EC)}$$

[4]É interessante notar que a teoria que desenvolveremos nesta seção é bem parecida à teoria das equações diferenciais lineares de ordem r, a coeficientes constantes.

O Polinômio

$$\mathcal{P}(\lambda) = \lambda^r - C_1\lambda^{r-1} - C_2\lambda^{r-2} - ... - C_r\lambda^0 \qquad \text{(PC)}$$

é dito **Polinômio característico** da recorrência (LCCH).

Se λ é uma raiz real de (EC) então deduzimos (e podemos verificar diretamente) que $a_n = \lambda^n$ é solução de (LCCH), além disso, pelo Princ. de sobrep, qualquer múltiplo de λ^n também é solução.

Se (EC) possui mais raízes reais, então cada uma produz uma família de soluções e combinações lineares delas também são soluções.

Exemplo 5.15. Para a recorr. $a_n = 8a_{n-1}$, a (EC) é $\lambda = 8$, logo $A8^n$ são soluções para todo $A \in \mathbb{R}$.

Para a recorr. $a_n = 5a_{n-1} - 6a_{n-2}$, a (EC) é $\lambda^2 - 5\lambda + 6 = 0$, cujas raízes são $\lambda = 2, 3$, logo $A2^n + B3^n$ são soluções para todo $A, B \in \mathbb{R}$. ★

Se λ é uma raiz real de (EC), de multiplicidade μ, então pode-se mostrar que λ^n, $n\lambda^n, ...,\ n^{\mu-1}\lambda^n$ são todas soluções de (LCCH) (e também combinações lineares delas).

Vejamos como exemplo ilustrativo o caso de raiz dupla. Note que se λ é raiz dupla de $\mathcal{P}(x)$ então $\mathcal{P}(x) = (x-\lambda)^2\mathcal{Q}(x)$ e logo $\mathcal{P}'(x) = 2(x-\lambda)\mathcal{Q}(x) + (x-\lambda)^2\mathcal{Q}'(x)$, ou seja, além de termos $\mathcal{P}(\lambda) = 0$ temos também $\mathcal{P}'(\lambda) = 0$.

Substituindo $n\lambda^n$ em (LCCH) temos

$$n\lambda^n = C_1(n-1)\lambda^{n-1} + C_2(n-2)\lambda^{n-2} + ... + C_r(n-r)\lambda^{n-r} = \sum_{i=1}^{r} C_i(n-i)\lambda^{n-i}\,,$$

que pode ser reescrito como

$$(n-r)\lambda^n + r\lambda^n = \sum_{i=1}^{r} C_i(n-r)\lambda^{n-i} + C_i(r-i)\lambda^{n-i}$$

e logo

$$\lambda^{n-r}\left\{(n-r)\left[\lambda^r - \sum_{i=1}^{r} C_i\lambda^{r-i}\right] + \lambda\left[r\lambda^{r-1} - \sum_{i=1}^{r}(r-1-i)\lambda^{r-i}\right]\right\} = 0\,,$$

que é de fato verdadeira pois os termos em colchete correspondem, respectivamente, a $\mathcal{P}(\lambda)$ e $\mathcal{P}'(\lambda)$.

Uma prova análoga pode ser usada para raízes de multiplicidade maior, observando que a derivada k-ésima do polinômio em λ é zero se k é menor (estrito) da multiplicidade da raiz λ.

Exemplo 5.16. Para a recorr. $a_n = 2a_{n-1} - a_{n-2}$, a (EC) é $\lambda^2 - 2\lambda + 1 = 0$, cuja raiz é $\lambda = 1$ de multiplicidade 2, logo $A1^n + Bn1^n = A + Bn$ são soluções para todo $A, B \in \mathbb{R}$. ★

Se $\lambda, \overline{\lambda}$ são duas raízes complexas conjugadas de (EC) então (todo o raciocínio feito até aqui poderia ser feito com $\mathbb{C}$ no lugar de $\mathbb{R}$), λ^n e $\overline{\lambda}^n$, junto com suas combinações lineares (até com coef. complexos) são soluções.

Como estamos procurando soluções reais, sejam $\rho = |\lambda|,\ \theta = arg(\lambda)$, assim

$$\lambda^n = \rho^n \left(\cos(n\theta) + i \sin(n\theta)\right),$$

$$\overline{\lambda}^n = \rho^n \left(\cos(n\theta) - i \sin(n\theta)\right)$$

são soluções e as combinações (reais)

$$(\lambda^n + \overline{\lambda}^n)/2 = \rho^n \cos(n\theta),$$

$$(\lambda^n - \overline{\lambda}^n)/2i = \rho^n \sin(n\theta)$$

são também soluções. Encontramos então duas soluções em correspondência das raízes conjugadas $\lambda, \overline{\lambda}$.

Enfim, como no caso de raízes reais múltiplas, se $\lambda, \overline{\lambda}$ são duas raízes de (EC) complexas conjugadas e de multiplicidade μ então

$$\rho^n \cos(n\theta),\quad \rho^n \sin(n\theta), ...,\ n^{\mu-1}\rho^n \cos(n\theta),\quad n^{\mu-1}\rho^n \sin(n\theta)$$

são soluções de (LCCH), onde $\rho = |\lambda|,\ \theta = arg(\lambda)$.

Resumindo o acima, temos

- se λ é uma raiz real de (EC) então

  $$\lambda^n \qquad \text{é solução de (LCCH)}$$

- se λ é uma raiz real de (EC), de multiplicidade μ, então

  $$\lambda^n,\ n\lambda^n, ...,\ n^{\mu-1}\lambda^n \qquad \text{são soluções de (LCCH),}$$

- se $\lambda, \overline{\lambda}$ são duas raízes complexas conjugadas de (EC) então

  $$\rho^n \cos(n\theta),\quad \rho^n \sin(n\theta) \qquad \text{são soluções de (LCCH),}$$

  onde $\rho = |\lambda|,\ \theta = arg(\lambda)$.

- se $\lambda, \overline{\lambda}$ são duas raízes de (EC) complexas conjugadas e de multiplicidade μ então

  $$\rho^n \cos(n\theta),\quad \rho^n \sin(n\theta), ...,\ n^{\mu-1}\rho^n \cos(n\theta),\quad n^{\mu-1}\rho^n \sin(n\theta)$$

  são soluções de (LCCH), onde $\rho = |\lambda|,\ \theta = arg(\lambda)$.

Observe que desta forma, em vista do Teorema Fundamental da Álgebra, obtemos sempre r diferentes[5] soluções. Para terminar a resolução completa da equação (LCCH), precisamos do resultado abaixo, cuja prova precisa de algumas noções de álgebra linear[6]:

Teorema 5.17. *Qualquer solução de* (LCCH) *pode ser escrita como combinação linear das r soluções obtidas.*

Exemplo 5.18.

- $a_n = \alpha a_{n-1}$

  é uma recorrência de ordem 1, logo a eq. característica é $\lambda = \alpha$, nos dando a solução $a_n = \alpha^n$. O espaço de todas as soluções é então $A\alpha^n : \ A \in \mathbb{R}.$ [7]

- $a_n = 5a_{n-1} - 6a_{n-2}$

  é uma recorrência de ordem 2, logo a eq. característica é $\lambda^2 = 5\lambda - 6$ e o polin. característico é $\lambda^2 - 5\lambda + 6$, nos dando as raízes $\lambda = 2, 3$ e as duas soluções $a_n = 2^n$ e $b_n = 3^n$. O espaço de todas as soluções é então $A2^n + B3^n : \ A, B \in \mathbb{R}.$

- $a_n = 2a_{n-1} - a_{n-2}$

  é uma recorrência de ordem 2, logo a eq. característica é $\lambda^2 = 2\lambda - 1$ e o polin. característico é $\lambda^2 - 2\lambda + 1$, nos dando a raiz dupla $\lambda = 1$, logo duas soluções (independentes) são $a_n = 1$ e $b_n = n$. O espaço de todas as soluções é então $A + Bn : \ A, B \in \mathbb{R}.$ [8]

- $a_n = -a_{n-2}$

  é uma recorrência de ordem 2, logo a eq. característica é $\lambda^2 = -1$ e o polin. característico é $\lambda^2 + 1$, nos dando as raízes complexas conjugadas $\lambda = \pm i$, logo duas soluções (independentes) são $a_n = \cos(n\pi/2)$ e $b_n = \cos(n\pi/2)$. O espaço de todas as soluções é então $A\cos(n\pi/2) + B\sin(n\pi/2) : \ A, B \in \mathbb{R}.$

  Note que as duas sequências são na verdade $a_n = 0, -1, 0, 1, 0, -1, \ldots.$ e $b_n = 1, 0, -1, 0, 1, 0, -1, \ldots.$, nos dando a solução geral $A, B, -A, -B, A, B, \ldots.$

[5] O termo correto seria dizer que as r soluções obtidas são linearmente independentes.

[6] De fato, a prova consiste em mostrar que o conjunto das soluções é um espaço vetorial de dimensão r, logo coincide com o espaço das combinações lineares das r soluções (linearmente independentes) obtidas.

[7] Compare com o exemplo 5.9.

[8] Compare com o exemplo 5.10.

- $a_n = a_{n-1} + a_{n-2}$

  é uma recorrência de ordem 2, logo a eq. característica é $\lambda^2 = \lambda + 1$ e o polin. característico é $\lambda^2 - \lambda - 1$, nos dando as raízes $\lambda = (1 \pm \sqrt{5})/2$ e as duas soluções $a_n = ((1+\sqrt{5})/2)^n$ e $b_n = ((1-\sqrt{5})/2)^n$. O espaço de todas as soluções é então

  $A((1+\sqrt{5})/2)^n + B((1-\sqrt{5})/2)^n : \ A, B \in \mathbb{R}.$

  Observe que pondo $a_1 = a_2 = 1$ obtemos a seq. de Fibonacci (veja no exemplo 5.11), que corresponde às constantes $A = -B = 1/\sqrt{5}$: apesar da formula contendo irracionais, a seq. obtida é composta apenas por inteiros. ★

5.3.2 Solução de recorrências lineares não homogêneas a coeficientes constantes

Vamos agora procurar uma forma de resolver certas recorrências não homogêneas, na forma (LCC).

Para isso usaremos o chamado **Método de semelhança**, que consiste em procurar uma solução que tenha forma parecida à própria $g(n)$.

Vamos considerar em particular os casos a seguir:

- **g polinômio de grau s**:
  procure uma solução em forma de polinômio:
  - de grau s se 1 não é raiz da (EC) da (LCCH),
  - de grau $s + \mu$ se 1 é raiz de multiplicidade μ da (EC) da (LCCH),
- **potência $\mathbf{g(n) = b^n}$**:
  procure uma solução na forma
  - Ab^n se b não é raiz da (EC) da (LCCH),
  - $An^\mu b^n$ se b é raiz de multiplicidade μ da (EC) da (LCCH).

Uma vez encontrada uma solução da (LCC) pelo método de semelhança, usando o Teorema 5.17 e o princípio de sobreposição, podemos escrever todas as soluções da (LCC), em particular vale o seguinte

Teorema 5.19. *Qualquer solução da* (LCC) *pode ser escrita como combinação linear das r soluções da homogênea associada* (LCCH)*, mais a solução da* (LCC) *obtida.*

Exemplo 5.20. • $a_n = 2a_{n-1} + n$

Como 1 não é raiz do polin. $\lambda - 2$ e $g(n) = n$ é um polin. de grau 1, procuro uma solução na forma $An + B$.

Substituindo temos $An+B = 2(A(n-1)+B)+n$, igualando os termos constantes e os termos em n temos o sistema

$$\begin{cases} A = 2A+1 \\ B = -2A+2B \end{cases}$$

e logo $A=-1,\ B=-2$, de fato $a_n = -n-2$ é uma solução.

Juntando com as soluções da homogênea associada, obtemos a solução geral (todas as possíveis soluções) $\boxed{a_n = C2^n - n - 2,\ A \in \mathbb{R}.}$

- $\boxed{a_n = a_{n-1} + n}$
  Como 1 é raiz simples do polin. $\lambda - 1$ e $g(n) = n$ é um polin. de grau 1, procuro uma solução na forma $An^2 + Bn + C$.

  Substituindo temos $An^2+Bn+C = (A(n-1)^2 + B(n-1) + C) + n$, igualando os termos constantes e os termos em n temos o sistema

$$\begin{cases} A = A \\ B = -2A + B + 1 \\ C = A - B + C \end{cases}$$

  e logo $A = B = 1/2$, de fato $a_n = n/2 + n^2/2$ é uma solução. [9]

  Juntando com as soluções da homogênea associada, obtemos a solução geral $\boxed{a_n = D + n/2 + n^2/2,\ D \in \mathbb{R}\,.}$

- $\boxed{a_n = a_{n-1} + 2^n}$
  Como 2 não é raiz do polin. $\lambda - 1$, procuro uma solução na forma $A2^n$.

  Substituindo temos $A2^n = (A2^{n-1}) + 2^n$, cortando 2^n temos a equação $A = A/2 + 1$ e logo $A = 2$, de fato $a_n = 2 \cdot 2^n$ é uma solução.

  Juntando com as soluções da homogênea associada, obtemos a solução geral (todas as possíveis soluções) $\boxed{a_n = C + 2 \cdot 2^n,\ C \in \mathbb{R}.}$

- $\boxed{a_n = 2a_{n-1} + 2^n}$
  Como 2 é raiz do polin. $\lambda - 2$, procuro uma solução na forma $An2^n$.

  Substituindo temos $An2^n = 2(A(n-1)2^{n-1}) + 2^n$, cortando 2^n temos o sistema

$$\begin{cases} A = 2A/2 \\ 0 = -A + 1 \end{cases}$$

[9] Note que C fica indeterminado, o que era de se esperar já que qualquer constante é solução da homogênea: poderíamos ter simplificado a conta procurando a solução já na forma $An^2 + Bn$.
Por outro lado, se esquecêssemos de pôr o termo An^2 no chute, a segunda equação do sistema seria $B = B + 1$, nos mostrando que o chute não está correto.

e logo $A = 1$, de fato $a_n = n2^n$ é uma solução.

Juntando com as soluções da homogênea associada, obtemos a solução geral (todas as possíveis soluções) $\boxed{a_n = C2^n + n2^n,\ C \in \mathbb{R}.}$ ★

Exercício 5.21.
- Resolva a recorr. $a_n = a_{n-1} + n2^n$ procurando uma solução na forma $An2^n + B2^n$ (obs, apenas um dos dois termos não daria certo!)
- Resolva a recorr. $a_n = a_{n-1} + 2^n + n2^n$ (use contas já feitas e o pr. de sobreposição).
- Resolva a recorr. $a_n = a_{n-1} + a_{n-2} + n$ ★

Exemplo 5.22 (Um caso útil)**.** Considere a recorrência $\boxed{a_k = \alpha\, a_{k-1} + \beta^k\,.}$

- A sol. geral da homogênea é $a_k = D\alpha^k, \qquad D \in \mathbb{R}$;
- uma sol. da eq completa é $a_k = \begin{cases} \frac{\beta}{\beta-\alpha}\,\beta^k & se\ \alpha \neq \beta, \\ k\beta^k & se\ \alpha = \beta; \end{cases}$
- a sol. geral é $\boxed{a_k = \begin{cases} D\alpha^k + \frac{\beta}{\beta-\alpha}\,\beta^k & se\ \alpha \neq \beta, \\ D\alpha^k + k\beta^k & se\ \alpha = \beta, \end{cases} \qquad D \in \mathbb{R}.}$
- Concluímos que a ordem da sol. geral é $\boxed{a_k = \begin{cases} \Theta(\alpha^k) & se\ \alpha > \beta\ (\text{e } D \neq 0), \\ \Theta(\beta^k) & se\ \alpha < \beta, \\ \Theta(k\alpha^k) & se\ \alpha = \beta. \end{cases}}$ ★

5.3.3 Lineares de primeira ordem

No caso a coeficientes não constantes, a resolução é mais complicada, mas podemos encontrar uma fórmula para a (LH) de primeira ordem, usando iteração. Considere (LH):

$$a_n = C_n a_{n-1}\,;$$

calculando alguns termos vemos que $a_1 = C_1 a_0$, $a_2 = C_2 a_1 = C_2 C_1 a_0$, $a_3 = C_3 a_2 = C_3 C_2 C_1 a_0, ...$, logo podemos deduzir

$$\boxed{a_n = \left[\prod_{i=1}^{n} C_i\right] a_0 \qquad \text{para } n > 0\,.} \tag{5.3}$$

Também podemos encontrar uma fórmula para a (L) de primeira ordem, usando iteração. Considere (L):

$$a_n = C_n a_{n-1} + f_n;$$

calculando alguns termos vemos que
$a_1 = C_1 a_0 + f_1$
$a_2 = C_2 a_1 + f_2 = C_2 C_1 a_0 + C_2 f_1 + f_2$,
$a_3 = C_3 a_2 + f_3 = C_3 C_2 C_1 a_0 + C_3 C_2 f_1 + C_3 f_2 + f_3$,
logo podemos deduzir [10]

$$\boxed{a_n = \left[\prod_{i=1}^{n} C_i\right] a_0 + \sum_{j=1}^{n} \left[\left(\prod_{i=j+1}^{n} C_i\right) f_j\right] .} \tag{5.4}$$

Note que na formula acima, o primeiro termo é a solução da homogênea associada (LH).

[10] Note a similaridade das fórmulas (5.3) e (5.4) com as soluções com condição inicial $y(0) = y_0$ dos problemas de Cauchy para a EDO linear homogênea $y'(t) = a(t)\, y(t)$, que é
$y(t) = e^{\int_0^t a(\tau) d\tau}\, y_0$,
e da EDO não homogênea $y'(t) = a(t)\, y(t) + f(t)$, que é
$y(t) = e^{\int_0^t a(\tau) d\tau}\, y_0 + \int_0^t \left(e^{\int_\sigma^t a(\tau) d\tau} f(\sigma)\right) d\sigma$.
A similaridade é maior ainda notando que, para coeficientes positivos, $\prod C_i = e^{\sum \ln(c_i)}$ e logo a (5.3) pode ser escrita como $a_n = e^{\sum \ln(c_i)}\, a_0$, enquanto a (5.4) pode ser escrita como $a_n = e^{\sum_{i=1}^{n} \ln(c_i)}\, a_0 + \sum_{j=1}^{n} \left(e^{\sum_{i=j+1}^{n} \ln(c_i)} f_j\right)$.

5.4 Recorrências que aparecem na análise de programas "divide e conquista"

Como já comentado, recorrências aparecem no estudo do peso computacional de programas de tipo "divide e conquista".

Este tipo de programa resolve um problema de tamanho n dividindo ele em sub-problemas menores e fazendo isso recursivamente até chegar num tamanho mínimo fácil de resolver, para no final juntar todos os problemas de novo e obter a solução final.

Exemplo 5.23.

- **Busca em lista ordenada.** Para encontrar um certo elemento numa lista ordenada, no lugar de correr a lista até achar o elemento (que pode ser muito rápido ou muito demorado, dependendo da posição do elemento buscado, mas em média precisa acessar a lista $n/2$ vezes), podemos dividir a lista em duas partes e, avaliando o elemento que divide as duas metades, limitar a busca à metade que deve conter o elemento buscado. Repetindo isso até chegar a uma lista de apenas um elemento, teremos encontrado ele, em cerca de $\ln_2(n)$ passos.

- **Ordenar lista.** Para ordenar uma lista existem muitos algoritmos, alguns muito pesados. Um algoritmo de tipo divide e conquista consiste em dividir a lista em duas partes recursivamente até chegar a listas de tamanho 1 (que serão então automaticamente ordenadas). Ao reunir os problemas precisará, em cada passo da volta da recursão, reunir duas listas (já ordenadas) em uma nova lista ordenada. Neste caso o maior trabalho é feito na hora da volta da recursão, mas como juntar duas listas já ordenadas é muito menos pesado do que ordenar uma lista qualquer, o método será melhor, pelo menos para listas grandes.

Generalizando os exemplos, em cada passo de um algoritmo deste tipo dividiremos o problemas em 'a' sub-problemas, sendo cada um de tamanho dividido por 'b', além disso, em cada passo teremos um trabalho adicional para fazer em cada sub-problema (proporcional ao seu tamanho) para a operação de divisão e eventualmente as operações de rejuntar os problemas na volta.

Podemos expressar isso com a recorrência abaixo, para o peso computacional T (trabalho) da resolução do problema de tamanho n (assumimos por enquanto que n seja uma potência de b):

$$\begin{cases} T(n) = aT(n/b) + f(n) \quad \textit{se } n > 1\,, \\ T(1) = t_1\,. \end{cases} \tag{DC}$$

A recorrência diz que

- para resolver um problema de tamanho n precisamos resolver a problemas de tamanho n/b mais um trabalho adicional $f(n)$.

- Para resolver um problema de tamanho 1 precisamos um trabalho t_1.

5.4.1 Método de solução usando recorrências de ordem 1

Se no problema (DC) fazemos as mudanças de variável e de sequência

$$n = b^k \iff \log_b n = k, \qquad A_k := T(b^k)\,,$$

obtemos o novo problema

$$\begin{cases} A_k = a\,A_{k-1} + f(b^k)\,, \\ A_0 = t_1\,. \end{cases} \tag{R1}$$

Considerando o caso $f(n) = n^c$ (tipicamente o trabalho adicional em cada passo será diretamente proporcional ao tamanho, aqui estamos assumindo uma relação expressa por uma potência de expoente fixado c), temos

$$\begin{cases} A_k = a\,A_{k-1} + (b^c)^k\,, \\ A_0 = t_1\,, \end{cases} \tag{R2}$$

cuja solução pode ser obtida como no exemplo 5.22:

$$A_k = \begin{cases} Da^k + \frac{b^c}{b^c-a}\,(b^c)^k & se\ a \neq b^c, \\ Da^k + k(b^c)^k & se\ a = (b^c), \end{cases} \qquad D \in \mathbb{R};$$

a constante D é determinada impondo $D + \frac{b^c}{b^c-a} = t_1$ (resp $D = t_1$), obtendo enfim

$$A_k = \begin{cases} \left(t_1 - \frac{b^c}{b^c-a}\right) a^k + \frac{b^c}{b^c-a}\,(b^c)^k & se\ a \neq b^c, \\ t_1\,a^k + k(b^c)^k & se\ a = (b^c)\,. \end{cases} \tag{5.5}$$

Em particular a ordem da sol. é dado por [11]

$$A_k = \begin{cases} \Theta(a^k) & se\ a > (b^c), \\ \Theta((b^c)^k) & se\ a < (b^c), \\ \Theta(ka^k) & se\ a = (b^c). \end{cases} \tag{5.6}$$

5.4.2 Método de solução usando árvore de recursão

No livro [SDK15a], para resolver a recorrência (DC), é usado o método de escrever uma tabela (ou desenhar um grafo) que descreve o problema em cada nível (veja a tabela 5.1).

A tabela mostra o seguinte.

- No nível 1 temos 1 problema de tamanho n, o trabalho do nível (em cada problema) é $f(n)$ e logo o trabalho total do nível é também $f(n)$.

[11] Note que no caso $a > (b^c)$, a constante D determinada é sempre positiva, logo a ordem de crescimento é mesmo a^k .

nível	1	2	3	..	k	k+1
# de prob.	1	a	a^2	..	a^{k-1}	a^k
tam. de cada prob.	n	$\frac{n}{b}$	$\frac{n}{b^2}$	..	$\frac{n}{b^{k-1}}$	1
trab. por prob.	$f(n)$	$f\left(\frac{n}{b}\right)$	$f\left(\frac{n}{b^2}\right)$	..	$f\left(\frac{n}{b^{k-1}}\right)$	t_1
Trab total	$f(n)$	$af\left(\frac{n}{b}\right)$	$a^2 f\left(\frac{n}{b^2}\right)$	..	$a^{k-1} f\left(\frac{n}{b^{k-1}}\right)$	$a^k t_1$

Tabela 5.1: Tabela para resolução da recorrência (DC).

- No nível 2 o problema está dividido em a sub-problemas, cada um de tamanho n/b, o trabalho do nível (em cada problema) é $f(n/b)$ e logo o trabalho total do nível é $af(n/b)$.
- Ao passar ao nível seguinte aplicamos a mesma divisão a cada problema do nível anterior, assim no nível j teremos a^{j-1} problemas, cada um de tamanho n/b^{j-1}, o trabalho do nível (em cada problema) é $f(n/b^{j-1})$ e logo o trabalho total do nível é $a^{j-1}f(n/b^{j-1})$.
- No nível $k+1$ (onde $n = b^k$) temos então a^k problemas, cada um de tamanho $n/b^k = 1$, logo o trabalho do nível (em cada problema) é agora t_1, pois chegamos no passo final do algorítimo, logo o trabalho total do nível é $a^k t_1$.

Somando o trabalho total de todos os níveis (última linha da tabela) obtemos

$$T(n) = \left[\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)\right] + a^k t_1 \,.$$

Se $f(n) = n^c$ como na seção anterior, usando as fórmula conhecidas para a soma de potências obtemos

$$\begin{aligned} T(n) &= \left[\sum_{i=0}^{k-1} a^i \left(\frac{n^c}{(b^c)^i}\right)\right] + a^k t_1 = n^c \frac{1-(a/b^c)^k}{1-(a/b^c)} + a^k t_1 \\ &= n^c \left(1-(a/b^c)^k\right) \frac{b^c}{b^c - a} + a^k t_1 \,, \end{aligned} \tag{5.7}$$

excepto no caso $a = b^c$ onde temos

$$T(n) = \left[\sum_{i=0}^{k-1} (n^c)\right] + a^k t_1 = k n^c + a^k t_1 \,. \tag{5.8}$$

Considerando que $n = b^k$, as fórmulas (5.7) e (5.8) coincidem com as em (5.5).

5.4.3 Teorema Mestre

Os dois métodos das seções 5.4.1 e 5.4.2 nos fornecem o chamado **Teorema Mestre**.

Teorema 5.24 (Teorema Mestre V1)**.**
Considere a recorrência (DC)

$$\begin{cases} T(n) = aT(n/b) + n^c \quad \textit{se } n > 1\,, \\ T(1) = t_1\,, \end{cases} \tag{5.9}$$

onde $a, c > 0$, $b > 1$, $t_1 \geq 0$ *e* n *é potência inteira de* b.
Então a ordem da sol. de (DC) *é*

- $T(n) = \Theta(a^{\log_b(n)}) = \Theta(n^{\log_b(a)})\,, \quad \textit{se } a > b^c\,,$
- $T(n) = \Theta(b^{c\log_b(n)}) = \Theta(n^c)\,, \quad \textit{se } a < b^c\,,$
- $T(n) = \Theta(\log_b(n)\, a^{\log_b(n)}) = \Theta(n^c \ln(n))\,, \quad \textit{se } a = b^c\,.$

Demonstração. As estimativas já foram obtidas na equação (5.6), apenas precisamos substituir $k = \log_b(n)$ e observar que:
– $a^{\log_b(n)} = b^{\log_b(a)\log_b(n)} = n^{\log_b(a)}$,
– $b^{c\log_b(n)} = n^{\log_b(b^c)} = n^c$,
– $\log_b(a) = c$ quando $a = b^c$. □

O teorema nos diz o seguinte:

- se $a > b^c$ o que mais pesa no algoritmo é o fato que estamos criando muito novos problemas (a estimativa depende de a, b),
- se $a < b^c$ o que mais pesa no algoritmo é o trabalho a ser feito em cada nível, (a estimativa depende de c),
- se $a = b^c$ os dois pesos acima estão balanceados (já que neste caso $c = \log_b(a)$).

5.4.4 Variações do Teorema Mestre

O Teorema Mestre na versão 5.24 possui algumas limitações:

- ele assume que o trabalho em cada passo é exatamente $f(n) = n^c$;
- ele assume que o tamanho do problema inicial é exatamente uma potência inteira de b.

Quando $f(n)$ não é conhecido precisamente, mas temos uma estimativa para ele, podemos usar os seguintes resultados.

Teorema 5.25 (Teorema Mestre V2)**.**
O Teorema Mestre vale também se n^c é substituído por $f(n)$ satisfazendo

$$f(n) = \Theta(n^c)\,.$$

Se

$$f(n) = O(n^c)$$

então o teorema vale substituindo Θ por O.

Em particular, se no lugar de uma informação precisa sobre $f(n)$ e t_1 temos apenas uma limitação para cima, temos

Teorema 5.26 (Teorema Mestre para inequações)**.**
Considere uma sequência tal que

$$\begin{cases} T(n) \leq aT(n/b) + n^c \quad \textit{se } n > 1\,, \\ T(1) \leq t_1\,, \end{cases} \tag{DC-ineq}$$

onde $a, c > 0$, $b > 1$ e $t_1 \geq 0$ e n é potência inteira de b.
Então

- $T(n) = O(n^{\log_b a})\ , \quad \textit{se } a > b^c,$
- $T(n) = O(n^c)\ , \quad \textit{se } a < b^c,$
- $T(n) = O(n^c \log n)\ , \quad \textit{se } a = b^c.$

No caso em que o tamanho do problema inicial não é exatamente uma potência inteira de b, ainda é possível fazer um algoritmo de tipo divide e conquista, mas em geral em cada subdivisão dos problemas, no lugar de obter problemas iguais de tamanho n/b, teremos problemas de tamanho variável, entre $\lfloor n/b \rfloor$ e $\lceil n/b \rceil$. [12]

Neste caso podemos escrever que

$$aT(\lfloor n/b \rfloor) + f(n) \leq T(n) \leq aT(\lceil n/b \rceil) + f(n)$$

e usar o seguinte resultado.

Teorema 5.27 (Teorema Mestre V3)**.**
O Teorema Mestre vale também se a recorrência é substituída por

$$aT(\lfloor n/b \rfloor) + f(n) \leq T(n) \leq aT(\lceil n/b \rceil) + f(n)\,,$$

onde $b \geq 2$ e n não precisa ser potência inteira de b.

[12] Aqui $\lfloor x \rfloor$ e $\lceil x \rceil$ são os arredondamentos de x ao inteiro menor (res. maior).

5.4.5 Comparação para recorrências e provas das variações do Teorema Mestre

Para mostrar os Teoremas 5.25 e 5.26, vamos estudar como comparar entre sí soluções de diferentes recorrências.

Teorema 5.28. *Considere as recorrências*

$$\begin{cases} a_n = C_1(n)a_{n-1} + ... + C_r(n)a_{n-r} + g(n)\,, \\ b_n = D_1(n)b_{n-1} + ... + D_r(n)b_{n-r} + h(n)\,. \end{cases}$$

Se

$$\begin{cases} 0 \le D_i(n) \le C_i(n) & \text{para } i = 1, ..., r \text{ e } n \ge n_0, \\ 0 \le h(n) \le g(n) & \text{para todo } n \ge n_0, \\ 0 \le b_{n_0-i} \le a_{n_0-i} & \text{para } i = 1, ..., r, \end{cases}$$

então $0 \le b_n \le a_n$ *para todo* $n \ge n_0$.

Corolário 5.29. *Considere a recorrência*

$$a_n = C_1(n)a_{n-1} + ... + C_r(n)a_{n-r} + g(n)$$

onde

$$\begin{cases} C_i(n) \ge 0 & \text{para } i = 1, ..., r \text{ e } n \ge n_0, \\ g(n) \ge 0 & \text{para } n \ge n_0, \\ a_{n_0-i} \ge 0 & \text{para } i = 1, ..., r. \end{cases}$$

Suponha que a sequência b_n *satisfaz*

$$\begin{cases} b_n \le C_1(n)b_{n-1} + ... + C_r(n)b_{n-r} + g(n) & \text{para todo } n \ge n_0, \\ b_{n_0-i} \le a_{n_0-i} & \text{para } i = 1, ..., r\,, \end{cases}$$

então $b_n \le a_n$ *para* $n \ge n_0$.

Prova das duas comparações. Nas hipóteses dadas, é fácil ver que pondo $n = n_0$ nas duas equações resulta $0 \le b_{n_0} \le a_{n_0}$. Procedendo por indução obtemos o resultado para todo $n \ge n_n$. □

Prova dos Teoremas 5.25 e 5.26. Transformando a recorrência (DC) como na seção 5.4.1, obtemos

$$\begin{cases} A_k = a\,A_{k-1} + f(b^k)\,, \\ A_0 = t_1\,; \end{cases} \tag{5.10}$$

considremos também a recorrência

$$\begin{cases} S_k = a\,S_{k-1} + (b^c)^k\,, \\ S_0 = 1\,. \end{cases} \tag{5.11}$$

Observe que se S_k satisfaz (5.11) então $\widetilde{S}_k = \xi S_k$ satisfaz

$$\begin{cases} \widetilde{S}_k = a\,\widetilde{S}_{k-1} + \xi (b^c)^k\,, \\ S_0 = \xi\,. \end{cases} \tag{5.12}$$

Como $f(n) = \Theta(n^c)$, existem constantes $R > r > 0$ tais que, para k grande, $r(b^c)^k \leq f(b^k) \leq r(b^c)^k$, aplicando então o Teorema 5.28 obtemos que $rS_k \leq A_k \leq RS_k$ para k grande, e como as estimativas para S_k são dadas pelo Teorema Mestre, elas valem também para A_k.

No caso de ter apenas a estimativa de tipo O-grande para f, podemos ainda obter a estimativa de tipo O-grande para a recorrência.

No caso do Teorema 5.26 podemos usar o mesmo raciocínio aplicando o Corolário 5.29. □

A prova do Teorema 5.27 é um pouco mais complicada, mas pode ser feita usando técnicas parecidas às usadas nos dois teoremas anteriores.

Referências Bibliográficas

[Cou00] S. C. Coutinho, *Números inteiros e criptografia RSA*, second ed., Série de Computação e Matemática [Computer and Mathematics Series], vol. 2, Instituto de Matemática Pura e Aplicada (IMPA), Rio de Janeiro; Sociedade Brasileira de Matemática, Rio de Janeiro, 2000. Cited at page(s) 79, 83

[SDK11] C STEIN, R DRYSDALE, and BOGART K., *Discrete mathematics for computer scientists.*, 1st edition ed., Pearson, 2011. Cited at page(s) 40

[SDK15a] C STEIN, R DRYSDALE, and BOGART K., *Matemática discreta - para ciência da computação.*, 1ª edição ed., In [SDK15b], 2015, Versão original: "Discrete Mathematics for Computer Scientists", 2011. Cited at page(s) 7, 12, 30, 32, 55, 109

[SDK15b] C STEIN, R DRYSDALE, and BOGART K., *Matemática discreta - para ciência da computação.*, 1ª edição ed., Pearson, 2015. Cited at page(s) 40, 115

Índice Remissivo