Introdução à Complexidade de Kolmogorov

Armando Matos

Nelma Moreira

Departamento de Ciência de Computadores

Faculdade de Ciências, Universidade do Porto

email: {acm,nam}@ncc.up.pt

Conteúdo

Bibliografia [LV90], [Cha87], [Cha94], [LV94]

Quanto é possível comprimir a informação dum objecto? Consideremos a unidade de informação o *bit*. A cada bit está associada uma escolha entre duas possibilidades e (a informação de) cada objecto pode ser codificado por uma sequência de bits. Assim, podemos restringir-nos ao estudo de sequências finitas de bits, isto é, a elementos de $\{0,1\}^\star$. Certamente, numa dada sequência pode haver bits em redundantes. Isto é, a mesma sequência pode ser descrita por outra de menor comprimento. Iremos estudar a possibilidade de *comprimir* a informação associada a um objecto.

Capítulo 1

Informação e Objectos

A Teoria de Informação segundo Shannon ([Sha48]) quantifica a informação duma situação em termos da probabilidade de cada uma das suas n instâncias.

Seja $C_n \subset \{0,1\}^\star$ o conjunto das sequências de comprimento n e suponhamos que cada uma é igualmente provável. Como $|C_n| = 2^n$ e a probabilidade de cada uma das suas sequências é $\frac{1}{2^n}$, a entropia associada é n. Isto é, o número de bits necessários para descrever uma qualquer sequência de C_n é precisamente igual ao seu comprimento. Contudo, existem sequências nesse conjunto que podem ser descritas com menos bits. Por exemplo, a sequência 1^n pode ser descrita em $O(\log n)$ bits, considerando a representação em binário de n e um algoritmo apropriado. Exemplo dum tal algoritmo é:

```
para i<-1 até n faça {
   escreva(1)
}
```

Esta sequência é *compressível* porque pode ser descrita com menos bits que o seu tamanho; diz-se que a informação associada é "pequena".

Por outro lado, uma sequência específica correspondente a n lançamentos duma moeda, associando "coroa" a 0 e "cara" a 1 provavelmente não se pode descrever com menos de n bits.

Consoante ao seu grau de regularidade ou, inversamente, de *aleatoriedade* uma sequência pode ser mais ou menos compressível.

Informalmente, a quantidade de informação (complexidade) duma sequência x é definida como o tamanho (número de bits) do menor "algoritmo" que tenha como resultado x.

A Complexidade de Kolmogorov (ou Teoria Algorítmica da Informação), tem como objectivo estudar a quantidade de informação dum objecto (sequência). A noção básica é a de aleatoriedade.

Uma sequência é aleatória se não for compressível. Iremos ver que, existem algumas sequências que são muito compressíveis, muitas outras que são incompressíveis e algumas que para serem comprimidas necessitam de algoritmos intratáveis.

Consideremos mais alguns exemplos.

(i) A sequência

$$110011001100\ldots1100 \quad = \quad (1100)^{1000}$$

é compressível, isto é, existe um programa p cujo resultado é essa sequência e cujo tamanho é menor que 4000 bits.

Exercício 1.0.1 *Escreva em linguagem C um programa que tenha como resultado a sequência* $(1100)^{1000}$*. Indique o tamanho do programa em bits .*

(ii) A sequência $\langle 3141596\cdots\rangle$ constituída pelos primeiros 10^6 dígitos de π, é compressível. Existem programas "pequenos" que produzem sucessivamente os dígitos de π. Esta sequência pode ser descrita em $O(1)$ bits.

(iii) Um milhão de bits obtidos por um gerador de pseudo-aleatórios. (função `rand`). Existem programas "pequenos" que geram esta sequência.

Critério Uma sequência $x \in \{0,1\}^\star$ é compressível se, numa linguagem fixada, existe um programa $p \in \{0,1\}^\star$ tal que p produz x e $|p| < |x|$.

Programas e máquinas de Turing

A linguagem ou modelo de computação que iremos considerar é a máquina de Turing. Em particular, o "programa" é o conteúdo inicial da fita de entrada de uma máquina de Turing universal específica.

Todas as máquinas de Turing que consideraremos têm 3 fitas e as correspondentes cabeças:

- Uma fita de entrada que contém o programa e que apenas pode ser lida, avançando a cabeça para a direita (de uma unidade).
- Uma fita de saída, inicialmente vazia, que apenas pode ser escrita, avançando a correspondente cabeça para a direita (de uma unidade).
- Uma fita de trabalho, inicialmente vazia, que pode ser lida e escrita, podendo a correspondente cabeça mover-se em cada passo de uma unidade para a direita ou para a esquerda.

As máquinas que consideramos são determinísticas, podendo cada passo ser uma operação de leitura, de escrita ou uma operação na fita de trabalho. Estas convenções não são essenciais; poderia existir apenas uma fita de trabalho cujo conteúdo inicial seria o programa. Contudo, são muito utilizadas e úteis em certas circunstâncias.

Se n é o número da máquina M, então, a máquina de Turing universal que tomamos como referência, com o programa 0^n1p, tem um comportamento idêntico ao da máquina M com o programa p. A sequência 0^n1p codifica o par de palavras $\langle n, p\rangle$; por exemplo, se $n = 1000$ e $p = 0011101'$, o par$\langle n, p\rangle$ fica codificado na palavra (dado que n é 8)

$$0^81p = 0000000010011101$$

Veremos outros processos de codificar um par de palavras x e y numa só palavra w (por forma que x e y possam ser calculados a partir de w). É óbvio que a simples concatenação xy não serve para esse efeito!

Capítulo 2

Complexidade Algorítmica

Seja $\Sigma = \{0,1\}$. Iremos identificar $\Sigma^\star$ com $\mathbb{N}$ considerando a bijecção que a cada $n \in \mathbb{N}$ faz corresponder a n-ésima sequência binária na ordem lexicográfica (crescente): $(0,\epsilon)$, $(1,0)$, $(2,1)$, $(3,00)$, $(4,01)$, $(5,10)$, $(6,11)$, $(7,000)$, ... Assim x pode denotar um inteiro ou uma sequência de bits. O comprimento ou tamanho (número de bits) de x é dado, como habitualmente, por $|x|$.

Exercício 2.0.2 *Determine formas fechadas para a bijecção indicada $f : \mathbb{N} \to \Sigma^\star$ e para a sua inversa f^{-1}.*

Seja M uma máquina de Turing calculadora de funções e $x,\ p \in \Sigma^\star$. Denotamos por $M(p) = x$ o facto da máquina de Turing M com dados p parar com a sequência x na fita (M calcula x com dados p). A complexidade $K_M(x)$ de x em relação a M é definida por:

$$K_M(x) \quad = \quad \begin{cases} min\{|p| \ : \ M(p) = x\} & \\ \infty & \text{se não existe } p \text{ tal que } M(p) = x \end{cases}$$

Exemplo 1 *Consideremos algumas máquinas de Turing e analisemos a complexidade de algumas sequência em relação a essas máquinas.*

(i) Seja M_1 uma máquina de Turing que se limita a copiar o programa x para a fita de saída. Então, se $x \in \Sigma^\star$ tem-se $K_{M_1}(x) = |x|$.

(ii) Seja M_2 uma máquina de Turing que para quaisquer dados apaga a fita e pára. Então

$$K_{M_2}(x) \quad = \quad \begin{cases} 0 & se\ x = \epsilon \\ \infty & se\ x \neq \epsilon \end{cases}$$

(iii) Seja M_3 uma máquina de Turing que calcule $2x$.

$$K_{M_3}(x) \quad = \quad \begin{cases} |x/2| & se\ x\ é\ par \\ \infty & caso\ contrário \end{cases}$$

(iv) Máquina Universal U. Se existe uma máquina de Turing M que, com um programa p calcula x então x também pode ser calculado pela máquina universal. Mais concretamente, a máquina u com o programa $0^{e(M)}1p$ produz x. Usando a tese de Church-Turing podemos afirmar que, se x pode ser calculado, então pode ser calculado por U. "Universal" é um nome bem posto!

Pode-se definir a complexidade de x em relação a outra sequência y. A ideia é que a informação de y pode ser usada para definir um programa p que calcule x. A complexidade $K_M(x|y)$ de x em relação a M condicional a y é definida por

$$K_M(x|y) \quad = \quad \begin{cases} min\{|p| \ : \ M(\langle p, y\rangle) = x\} & \\ \infty & \text{se não existe } p \text{ tal que } M(\langle p, y\rangle) = x \end{cases}$$

onde $\langle \cdot, \cdot \rangle$ é a bijecção usual de $\mathbb{N}^2$ em $\mathbb{N}$.

Exemplo 2 *Para qualquer máquina de Turing M e para toda a sequência $x \in \Sigma^\star$, tem-se que $K_M(x|\epsilon)$ é igual a $K_M(x)$. E como será $K_M(x|x)$?*

Pela definição, a complexidade K_M de x depende da máquina M, portanto a noção de compressibilidade é também dependente da máquina usada. Em particular, uma sequência x pode ser incompressível (aleatória) para uma máquina M e compressível para outra máquina M'. Vamos ver contudo que, a menos duma constante, a complexidade de x não depende da máquina M escolhida. O Teorema da Invariância permite concluir que dadas duas máquinas de Turing M_1 e M_2 para todo $x \in \Sigma$

$$|K_{M_1}(x) - K_{M_2}(x)| \leq c_{M_1 M_2}$$

isto é, a complexidade de x em relação a duas máquinas difere duma constante que não depende de x mas apenas das máquinas referidas.

Teorema 2.0.1 (Teorema da Invariância) *Existe uma máquina de Turing U, tal que para toda a máquina de Turing M, existe uma constante c_M tal que para toda a sequência $x \in \Sigma^\star$*

$$K_U(x) \leq K_M(x) + c_M$$

Dem. Seja U uma máquina universal de Turing e seja $n = e(M)$ o número da máquina M para uma dada enumeração (codificação) e. Suponhamos que $M(p) = x$. Considerando os inteiros codificados em unário temos que $U(O^n 1p) = x$. Escolhendo $c_M = n + 1$ vem

$$\forall x \in \{0,1\}^\star \ \ K_U(x) \leq n + 1 + K_M(x)$$

□

O Teorema da Invariância também é válido se considerarmos a complexidade $K_M(x|y)$ de x condicional a y.

Exercício 2.0.3 Enuncie e prove o Teorema da Invariância para complexidades condicionais.

O Teorema da Invariância pode ainda ser expresso em termos de funções recursivas.

Exercício 2.0.4 Defina complexidade de $x \in \Sigma^\star$ em relação a uma função recursiva f.

Teorema 2.0.2 (Teorema da Invariância para funções recursivas) *Existe uma função recursiva f_0 tal que para toda a função recursiva f existe uma constante c_f tal que para toda a sequência $x \in \Sigma^\star$*

$$K_{f_0}(x) \leq K_f(x) + c_f$$

Em todos os casos, considerando sequências x suficientemente grandes a constante pode ser desprezada. Podemos, ainda, fixar uma máquina universal U de referência e estudar a complexidade duma sequência x em relação apenas a U. O índice U pode ser omitido na definição de $K(x)$ ou de $K(x|y)$

Exemplo 3 *Mostrar que para todo o $x \in \Sigma$, $K(xx) \leq K(x) + O(1)$.*

Seja U a máquina de referência, de número $n = e(U)$ e seja p o programa mínimo tal que U com dados p calcula x. Tem-se que $K(x) = |p|$. Seja V uma máquina de Turing, de número $n_V = e(V)$, que com dados 0^n1p simula U com dados p, mas que duplica o resultado antes de parar. Assim V com dados 0^n1p calcula xx e portanto U com dados $0^{n_V}10^n1p$ também calcula xx. Então, para todo o $x \in \Sigma^\star$

$$K(xx) \leq K(x) + n_V + n + 2$$

Exemplo 4 *Seja $K(x,y) = K(\langle x,y\rangle >)$, onde $\langle .,. >$ é a bijecção usual de $\mathbb{N}^2$ em $\mathbb{N}$. Poderíamos pensar que, para quaisquer x e y,*

$$K(x,y) \leq K(x) + K(y) + O(1)$$

Vamos ver que isso não é verdade.

Seja p o menor programa que produz x e q o menor programa que produz y. Suponhamos que M é uma máquina, de número $n = e(M)$, que dado pq usa p para produzir x e em seguida usa q para produzir y. Se M existir então U com dados 0^n1pq calcula $\langle x,y\rangle$ e ter-se-ia

$$K(x,y) \leq K(x) + K(y) + n + 1$$

Contudo, uma tal máquina não existe porque é necessário saber dividir os dados pq de modo a identificar p e q. Isto é, não se pode usar como codificação de p e q a sequência pq e portanto M não pode ser de complexidade constante.

Podemos separar p de q se for conhecido o comprimento de p, por exemplo, se os dados forem $|p|pq$ e soubermos separar p de $|p|$.

Podemos codificar $|p|$ em $O(\log |p|)$ bits considerando o seguinte método: dada uma sequência $x \in \Sigma^\star$, a sequência $\overline{x}$ é obtida inserindo um 0 entre dois símbolos adjacentes de x, e adicionando um 1 no final. Por exemplo, $\overline{01011} = 0010001011$. Com esta representação é imediato detectar quando uma sequência termina. A sequência $x' = \overline{|x|}x$ é denominada uma versão auto-delimitada *de x e tem comprimento $|x| + 2\log|x|$ (bits). Se x for a representação em binário de um inteiro n, então x' requer $\log n + 2\log\log n$ bits.*

Voltando ao nosso exemplo, suponhamos $p = 110101101$ e $q = 1110110111$. Então, $|p| = 9$ e $\overline{|p|} = \overline{1001} = 10000011$. A codificação de p e q é então

$$p'q = 100000111101011011110110111$$

No caso geral, tem-se que

$$K(\langle x, y\rangle) \leq K(x) + K(y) + O(\log min(K(x), K(y)))$$

Exemplo 5 *Linguagens* `C` *e* `Prolog`*. Será que um programa de manipulação simbólica em Prolog é menor que um programa equivalente em C? E para um programa de calculo numérico ter-se-á o inverso? O Teorema da Invariância mostra que para exprimir sucintamente um algoritmo num programa não interessa qual a linguagem de programação usada, a menos duma constante aditiva que apenas depende das duas linguagens e não do algoritmo. Seja $\pi_1, \pi_2, \ldots$ uma enumeração lexicográfica de todos os programas sintácticamente correctos em* `Prolog` *e $\gamma_1, \gamma_2, \ldots$ uma enumeração análoga para os programas em* `C`*. Cada programa de ambas as linguagens pode ser visto como uma função recursiva parcial dos seus dados nos seus resultados. Escolhendo programas de referência adequados podemos definir $K_{\texttt{Prolog}}(x)$ e $K_{\texttt{C}}(x)$ em analogia com $K(x)$. Todas esta medidas coincidem a menos de uma constante aditiva. Provemos este facto. Cada enumeração contém um programa universal; a enumeração de* `C` *contém um interpretador de C e também um interpretador de Prolog, seja γ_p. Então, $K_{\texttt{C}}(x) \leq K_{\texttt{Prolog}}(x) + c_p$, onde c_p é uma constante associada a p. Por exemplo, c_p pode ser $2|p| + 1$ se codificarmos p de forma auto-delimitada por $0^{|p|}1p$. Analogamente se prova que existe π_c tal que $K_{\texttt{Prolog}}(x) \leq K_{\texttt{C}}(x) + c_c$. Logo, para todo o x, $|K_{\texttt{C}}(x) - K_{\texttt{Prolog}}(x)| \leq c_p + c_c$*

Capítulo 3

Compressibilidade e Aleatoriedade

O Teorema da Invariância permite obter um majorante da complexidade $K(x)$ de $x \in \Sigma^\star$. Intuitivamente, a complexidade de Kolmogorov de x não deve exceder, a menos de uma constante, o seu comprimento $|x|$, dado que x é obviamente uma descrição de x.

Teorema 3.0.3 *Existe uma constante c, tal que para todo o x e y tem-se que*

$$\begin{aligned} K(x) &\leq |x| + c \\ K(x|y) &\leq K(x) + c \end{aligned}$$

Dem. Seja $n = |x|$ e M uma máquina de Turing que apenas copia os seus dados (programa) para a fita de saída. Para todo $x \in \Sigma^\star$, $K_M(x) = n$. Pelo Teorema da Invariância, $p = 0^{e(M)}1x$ é um programa para a máquina universal de referência U que produz x. Então vem,

$$K(x) \leq n + e(M) + 1$$

o que prova a primeira inequação.

Para provar a segunda inequação, constrói-se uma máquina de Turing M que para todos os $y,\ z \in \Sigma^\star$ calcula x com dados $\langle y, z\rangle$ se e só se a máquina universal de referência U calcula x dado z. Então, $K_M(x|y) = K(x)$. Pelo Teorema da Invariância para condicionais, existe c tal que

$$K(x|y) \leq K_M(x|y) + c = K(x) + c$$

□

Dado que temos um majorante da complexidade de x, será que podemos obter minorantes da complexidade de x? Quanto pode ser comprimida uma dada sequência x? Por outro lado, será que todas as sequências são compressíveis? Será que existem sequências incompressíveis? E quantas?

Já vimos que há sequências que podiam se descritas por programas muito mais curtos que elas próprias. Consideremos outro exemplo. Seja $f(1) = 2$ e $f(i) = 2^{f(i-1)}$ para todo $i > 1$. Esta função cresce muito rapidamente com i. Contudo para cada i o inteiro $y = 2^{f(i)}$ tem no máximo complexidade $K(i) + c$, para uma constante c independente de i.

E quanto à incompressibilidade? Dizemos que uma sequência x de tamanho $|x| = n$ é *incompressível* se $K(x) \geq n$.

Proposição 3.0.1 *Para todo o $n > 1$ existe uma sequência x de tamanho $|x| = n$ que é incompressível.*

Dem. Um simples argumento de contagem demonstra o resultado. Para qualquer $n \geq 0$ existem 2^n sequências binárias de tamanho n. Contudo o número de descrições (programas) de tamanho menor que n é apenas $\sum_{i=0}^{n-1} 2^i = 2^n - 1$.

□

Exercício 3.0.5 Mostre para qualquer n e para qualquer sequência $y \in \Sigma^\star$, existe uma sequência $x \in \Sigma^\star$, com $|x| = n$ tal que $K(x|y) \geq n$.

Exemplo 6 *Será que uma sequência incompressível pode ter uma subsequência compressível? Seja $x = uvw$ tal que x pode ser descrita por um programa pequeno p para v e pela sequência uw. Se para cada uma das subsequências p e u se considerar um prefixo com uma descrição auto-delimitada do seu tamanho, então $q = \overline{|p|}p\overline{|u|}uw$ é um programa para x. E, $|q| \leq K(v) + |uw| + O(\log |x|)$ bits. Então,*

$$K(x) \leq K(v) + |uw| + O(\log |x|)$$

Se x for incompressível é $K(x) \geq |x| + c$, logo temos (dado que $|x| = |v| + |uw|$):

$$K(v) \geq |v| - O(\log |x|)$$

Intuitivamente vemos que, se x e v forem de tamanhos comparáveis, v pode, em princípio, ser compressível mas muito pouco...

Quantas são as sequências que são incompressíveis? Existe pelo menos uma sequência de tamanho n que não pode ter uma complexidade menor que n. Pelo menos metade das sequências de tamanho n não são compressíveis para complexidade menor que $n-1$, uma vez que só existem $2^{n-1} - 1$ programas de tamanho menor que $n-1$. Pelo mesmo raciocínio, pelo menos três quartos de todas as sequências de tamanho n não podem ser compressíveis para complexidade menor que $n-2$; e assim por diante. Para $n-c$, só 2^{-c} sequências de tamanho n podem ser descritas com menos de $n-c$ bits.

Em geral, para cada constante c dizemos que uma sequência x é *c-incompressível* se $K(x) \geq |x| - c$.

Teorema 3.0.4 (Teorema da Incompressibilidade) *Seja c um inteiro positivo. Para cada y fixo, todo o conjunto finito $A \subseteq \Sigma^\star$ com cardinal $\#A = m$ tem pelo menos $m(1-2^{-c})+1$ elementos x com $C(x|y) \geq \log m - c$.*

Exercício 3.0.6 Mostre o teorema 3.0.4.

Sendo $A = \{x \ : \ |x| = n\}$ tem-se que $\#A = 2^n$. Pelos teoremas 3.0.4 e 3.0.3 tem-se que para quase todo o $x \in A$ x, $n - c \leq K(x) \leq n + c$ sendo c a constante do teorema 3.0.3. Isto é, há poucas sequências de complexidade baixa...

Exemplo 7 *Seja $p(x)$ o menor programa que produz $x \in \Sigma^\star$ na máquina de referência U, isto é, $K(x) = |p(x)|$. Vamos ver que $p(x)$ é incompressível, no sentido em que existe uma constante $c > 0$ tal que para toda a sequência x, $K(p(x)) \geq |p(x)| - c$.*

Por redução ao absurdo, suponhamos que para qualquer constante c existe um x e uma programa $q = p(p(x))$ que produz p e tal que $|q| < |p| - c$. Seja V uma máquina semelhante à máquina universal de referência U excepto que V simula U e depois usa o seu resultado como dados e simula U mais uma vez. Seja $n = e(V)$. Tem-se que U com dados $0^n 1q$ calcula x. Então, $K(x) < |p| - c + n + 1$ e se $c \geq n + 1$ obtemos uma contradição. Podemos concluir então que para $p(x)$ suficientemente grandes tem-se

$$\lim_{|p(x)|\to\infty} \frac{K(p(x))}{|p(x)|} = 1$$

Exemplo 8 *Quantas sequências $x \in \Sigma^\star$ têm $K(x) < |x| - 20$?*

Considere-se a razão

$$\frac{2^{|x|-20} - 1}{2^{|x|}} \approx 10^{-6}$$

Exemplo 9 *Quantas sequências $x \in \Sigma^\star$ têm $K(x) \leq \alpha|x|$ com $0 < \alpha < 1$?*

Considere-se a razão

$$f_\alpha(n) = \frac{\#\{x \ : \ |x| = n, \ K(x) \leq \alpha n\}}{\#\{x \ : \ |x| = n\}} \leq \frac{2^{\lfloor \alpha n \rfloor + 1} - 1}{2^n} < 2^{-(1-\alpha)n+1}$$

Então

$$\lim_{n\to\infty} f_\alpha(n) = 0$$

Do exemplo anterior, podemos concluir que quase todas as sequências são incompressíveis!!

Dada uma função $g(n)$ dizemos que uma sequência x de tamanho n é *g-incompressível* se

$$K(x) \geq n - g(n)$$

Como acima, há 2^n sequências de tamanho n e apenas $2^{n-g(n)}-1$ descrições de tamanho menor que $n-g(n)$. Então,

$$f(n) = \frac{\#\{x \ : \ |x| = n, \ K(x) < n - g(n)\}}{\#\{x \ : \ |x| = n\}} = \frac{2^{n-g(n)} - 1}{2^n} < 2^{-g(n)}$$

e $\lim_{g(n)\to\infty} f(n) = 0$

As sequências que são incompressíveis (ou c-incompressíveis para c pequeno) não apresentam regularidades, senão essa regularidade poderia ser usada para a comprimir. Intuitivamente, dizemos que essas sequências são *aleatórias*. Formalmente pode-se provar que as sequências incompressíveis verificam os testes clássicos de aleatoriedade.

Uma sequência finita x de tamanho n diz-se *aleatória (segundo Kolmogorov)* se é $O(\log n)$-incompressível.

Capítulo 4

Propriedades de K como função inteira

A função $K(x)$ pode ser vista como uma função de $\mathbb{N}$ em $\mathbb{N}$.

Lema 4.0.1 *Para qualquer $x \in \Sigma^\star$ tem-se que:*

a) $K(x) \leq |x|$ *a menos uma constante que não depende de* x

b) Para todo a constante c,

$$\frac{\#\{x \ : \ |x| = n,\, K(x) < n - c\}}{2^n} < 2^{-c}$$

c) $\lim_{x\to\infty} K(x) = \infty$

d) Seja $m(x) = min\{K(y) \ : \ y \geq x\}$*, isto é, a maior função monótona crescente que limita inferiormente* $K(x)$*. Então,* $\lim_{x\to\infty} m(x) = \infty$*.*

e) Para qualquer função parcial recursiva $\phi(x)$ *que tenda monotonamente para* ∞ *tem-se que* $m(x) < \phi(x)$*, para* x *suficientemente grande.*

f) $K(x)$ *é contínua no sentido em que para qualquer* h*,* $|K(x+h) - K(x)| \leq 2|h|$ *a menos duma constante que não depende de* h *nem de* x*.*

Dem. d) Para cada n existe um menor x_n tal que para todo o $x > x_n$ o menor programa p que produz x tem tamanho maior ou igual a n. Isto porque, para cada n existe apenas um número finito de programas de tamanho $< n$. Claramente, $x_{n+1} \geq x_n$, para todo o n. Por outro lado, $m(x) = n + 1$ para todo o $x_n < x < x_{n+1}$. O que prova o pretendido.

e) Por redução ao absurdo. Suponhamos que existia $\phi(x)$ tal que $\phi(x) \leq m(x)$ para uma infinidade de x. O conjunto $A = \{x \ : \ \phi(x) < \infty\}$ é infinito e recursivamente enumerável. Então, tem um subconjunto $B \subset A$ que é infinito e recursivo. Seja

$$\psi(x) \quad = \quad \begin{cases} \phi(x) & \text{se } x \in B \\ \phi(y) & \text{com } y = max\{z \ : \ z \in B, \ z < x\}, \text{ caso contrário} \end{cases}$$

A função ψ é total recursiva e $\psi(x) \leq m(x)$ para uma infinidade de x.

Defina-se $M(a) = max\{x \ : \ K(x) \leq a\}$. Então, $M(a) + 1 = min\{x \ : \ m(x) > a\}$. Tem-se que

$$F(a) = max\{x \ : \ \psi(x) \leq a + 1\} \geq min\{x \ : \ m(x) > a\} > M(a)$$

para uma infinidade de a's e $F(a)$ é recursiva total. Isto é, $K(F(a)) > a$ para uma infinidade de a's. Mas pelo Teorema da Invariância

$$K(F(a)) \leq K_F(F(a)) + O(1) \leq |a| + O(1)$$

Isto implica que existe uma constante c tal que $|a| + c \geq a$, para um número infinito de a's, o que é impossível.

f) Seja p o programa tal que $K(x) = |p|$. Podemos descrever $x + h$ por $\overline{h}p$. Como $|\overline{h}| \leq 2|h|$, tem-se o pretendido (a menos uma constante). □

Um efeito da quantidade de informação associada ao tamanho de uma sequência x é que $K(x)$ é não monótona *em prefixos*. Mais concretamente:

Exemplo 10 *É certo que, para $m \leq n$ podemos ter $K(m) > K(n)$ (Porquê?). Mas então, para $x = 1^n$ e $y = 1^m$ vem $K(x) < K(y)$, apesar de y ser um prefixo próprio de x. Por exemplo, se $n = 2^k$ então $K(1^n) \leq \log\log n + O(1)$, mas pelo Teorema da Incompressibilidade existe $m < n$ tal que $K(1^m) \geq logn - O(1)$. Em conclusão, a complexidade duma parte é maior que a complexidade do todo.*

Podemos tentar resolver a incongruência do exemplo anterior considerando o tamanho de x como dado, isto é, considerando a complexidade condicional de x dado $|x| = n$, $K(x|n)$. Informalmente isto permite um ganho da ordem de $\log|x|$ bits no tamanho do menor programa para x.

Para qualquer sequência x, com $|x| = n$ tem-se $K(x|n) \leq K(x) + O(1)$. No caso geral o tamanho de x não parece poder fornecer muita informação acerca do padrão de 0's e 1's em x. No entanto, às vezes a informação de $|x|$ permite determinar esse padrão:

Exemplo 11 *Seja $x = 1^n$. Então, $K(x) = K(n) + O(1)$ e $K(x|n) = O(1)$.*

Contudo, como se disse, no caso geral $K(x|n)$ não vai resolver o problema da não monotonia de prefixos:

Exemplo 12 *Seja $x = n0\ldots0$ com $|x| = n$, isto é, $x = n0^{n-|n|}$. Para cada n, x é denominada a n-sequência. Existe uma constante c tal que para todo o n, $K(x|n) \leq c$. Por exemplo, dado n podemos encontrar a n-ésima sequência binária (considerando a bijecção referida entre $\mathbb{N}$ e $\Sigma^\star$) e preencher com zeros até obter uma sequência de tamanho n. Suponhamos que n é aleatório, isto é, que $K(n) \geq |n|$. Então, para $x = n0^{n-|n|}$ tem-se ainda $K(x|n) \leq c$. Mas $K(n||n|) \geq K(n) - K(|n|) \geq \log n - 2\log\log n$.*

Teorema 4.0.5 *A função $K(x)$ não é uma função recursiva. Mais ainda, nenhuma função parcial recursiva $\phi(x)$, definida num conjunto infinito, pode coincidir com $K(x)$ em todo o seu domínio de definição.*

Dem. Qualquer conjunto infinito r.e. contém um subconjunto infinito recursivo. Seja A um conjunto recursivo infinito no domínio de definição de ϕ. A função $\psi(m) = min\{x : K(x) \geq m,\ x \in A\}$ é total recursiva (dado que $K(x) = \phi(x)$ em A) e toma valores arbitrariamente grandes. Também, por definição de ψ, $K(\psi(m)) \geq m$. Por outro lado, $K(\psi(m)) \leq K_\psi(\psi(m)) + c_\psi$ e $K_\psi(\psi(m)) \leq |m|$. Então, $m \leq \log m$ a menos uma constante independente de m, o que é falso. □

Logo, pela Tese de Church-Turing $K(x)$ não é computável.

Contudo,

Teorema 4.0.6 *Existe uma função recursiva total $\phi(t, x)$, monótona decrescente em t, tal que $\lim_{t\to\infty} \phi(t, x) = K(x)$.*

Dem. Seja c a constante tal que $K(x) \leq |x| + c$ para todo o x. Define-se, $\phi(t, x)$ como o tamanho do menor programa p, com $|p| \leq |x| + c$, e tal que a máquina de referência U com dados p pára com resultado x em t passos. Basta notar que o conjunto dos p tal que $|p| \leq |x| + c$ é finito e portanto podemos executar U para cada um desses p por t passos. Suponhamos primeiro que a máquina pára com resultado x para um ou mais desses p; neste caso define-se $\phi(t, x)$ como o menor comprimento desses programas. Se não existir p nessas condições, define-se $\phi(t, x)$ é $|x| + c$. Claramente, $\phi(t, x)$ é recursiva total e monotonamente decrescente com t. O limite existe pois para cada x existe um t tal que U pára com resultado x depois de t passos começando com dados p tal que $K(x) = |p|$. □

Embora se tenha provado que quase todas as sequências são aleatórias, num dado sistema formal apenas para um número finito dessas sequências é possível demonstrar a sua aleatoriedade.

De facto, para qualquer função recursiva total tal que $\lim_{x\to\infty} f(x) = \infty$ o conjunto dos x tal que se pode provar que $K(x) > f(x)$ é finito. Assim se $f(x) \ll |x|$, embora se saiba que quase todos os x verificam $K(x) \gg f(x)$ isso só se pode demonstrar para um número finito de x.

Teorema 4.0.7

(i) O conjunto $A = \{(x, a) : K(x) \leq a\}$ é r.e. mas não é recursivo.

(ii) Qualquer função parcial recursiva $\phi(x)$ que é um minorante de $K(x)$ é limitada (isto é, existe c tal que, para todo o x, é $\phi(x) < c$).

(iii) Seja $f(x)$ uma função total recursiva tal que $f(x) \leq \log x$ e $\lim_{x\to\infty} f(x) = \infty$. Então, o conjunto $B = \{x : K(x) \leq f(x)\}$ é simples*, isto é, é r.e. e o complementar de B é infinito mas não contém um subconjunto infinito r.e.*

Dem.

(i) O facto de A ser r.e. resulta do Teorema 4.0.6. Contudo, A não é recursivo. Sabemos que existe uma constante c tal que para todo o x, tem-se que $K(x) \leq |x| + c$. Se A fosse recursivo então usando este majorante podíamos calcular $K(x)$ (Verifique!), o que contradiz o Teorema 4.0.5.

(ii) Seja ϕ uma função parcial recursiva e seja $D = \{x : \phi(x) \leq K(x)\}$. Se D é finito nada há a provar. Suponhamos que D é infinito e ϕ é ilimitada. Considere-se uma enumeração do domínio de ϕ e defina-se a função total recursiva g

$$g(n) = min\{x : \phi(x) \geq n\}$$

onde o mínimo é considerado nessa enumeração. Supondo ϕ ilimitada, para cada n existe um tal x. Seja $k = e(\phi)$ na enumeração de todas as funções recursivas associada à máquina universal de referência U. Então, podemos usar ϕ e n para calcular x, isto é, $K(x) \leq |n| + |\overline{k}|$ a menos duma constante. Por outro lado, $K(x) \geq \phi(n) \geq n$. Para n suficientemente grande, obtemos uma contradição.

(iii) B é r.e. pela alínea (i). O complementar de B, $\overline{B} = \{x : K(x) > f(x)\}$ é infinito pelo Teorema da Incompressibilidade. Vamos demonstrar que B é simples. Suponhamos que existia D, conjunto infinito r.e. contido em $\overline{B}$. A restrição f_D de f a D é uma função parcial recursiva que é um limite inferior de $K(x)$. Pela alínea (ii) f_D é limitada. Como $f(x)$ é ilimitada isto implica que D é finito. Absurdo!

□

Teorias formais e complexidade de Kolmogorov

Recordemos sucintamente a noção de sistema (teoria) formal. Uma teoria formal T é um conjunto de fórmulas. Numa teoria T considera-se o conjunto das fórmulas *verdadeiras* (axiomas) e o conjunto de fórmulas *demonstráveis* em T, de acordo com uma noção (sintáctica) de demonstração. Uma teoria é *axiomatizável* se for r.e. Por exemplo, se os seus axiomas podem ser efectivamente enumerados e se existe um algoritmo que enumera todas as demonstrações de fórmulas de T a partir dos axiomas. Uma teoria é *decidível* se o seu conjunto é recursivo. Uma teoria é *consistente* se para toda a fórmula x, não é verdade que x e $\neg x$ estejam ambas em T. Uma teoria é *integra* se todas as fórmulas de T são verdadeiras. Por exemplo, os axiomas da Aritmética de Peano (considerados como uma teoria da lógica de 1ª ordem) são uma axiomatização da teoria elementar dos números.

O corolário seguinte é uma versão do Teorema da Incompletitude de Gödel, isto é, que um sistema formal que contenha a aritmética de Peano ou é inconsistente ou contém teoremas (fórmulas verdadeiras) que não podem ser demonstrados no sistema.

Corolário 4.0.1 *Existe um conjunto B r.e. com um complementar infinito, tal que para toda a teoria T consistente e axiomatizável existe apenas um número finito de n's tal que a fórmula $n \notin B$ é verdadeira e demonstrável em T. (Mas, com um número finito de excepções um número infinito dessas fórmulas são verdadeiras).*

Dem. Seja B o conjunto do Teorema 4.0.7, alínea (iii), e seja $\overline{B}$ o seu complementar. O conjunto $D \subseteq \overline{B}$ dos elementos n que possuem uma demonstração em T que pertencem a $\overline{B}$ é r.e. Como B é *simples*, $\overline{B}$ não contém um conjunto infinito r.e. Então, D é finito. □

Exemplo 13 (Chaitin) *Seja T uma teoria integra axiomatizável cujos axiomas e regras de inferência podem ser descritas com cerca de k bits. Então, T não pode ser usado para demonstrar a aleatoriedade de nenhum número com mais de k bits. Se o sistema pudesse demonstrar a aleatoriedade para um número com muito mais de k bits então a primeira dessas demonstrações (na enumeração de todas as demonstrações obtidas por aplicação repetida dos axiomas e regras de inferência) podia ser usada para obter uma contradição: um programa com aproximadamente k-bits que produzia o número aleatório, referido na demonstração, um número para o qual, por hipótese, o menor programa que o produzia tinha muito mais que k bits. Formalmente,*

- *Seja T uma teoria axiomatizável (isto é r.e.) que pode ser descrita com k bits: $K(T) \leq k$*
- *Suponhamos que todas as fórmulas de T são verdadeiras (no modelo dos número naturais)*
- *Seja $S_c(x)$ a fórmula com o significado*

x é lexicograficamente a menor sequência de tamanho c com $K(x) \geq c$

Tem-se que $K(S_c) \leq \log c$ *a menos uma constante fixa independente de* T *e de* c.

Para cada c, *existe um* x *tal que* $S_c(x) = \mathbf{true}$ *é uma afirmação verdadeira. E* S_c *garante que esse* x *é único. Combinando as descrições de* T *e de* S_c *obtemos uma descrição para* x. *Nomeadamente, para cada candidato* y *de tamanho* c, *podemos decidir se* $S_c(y) = \mathbf{true}$ *ou se* $\neg S_c(y) = \mathbf{true}$ *por enumeração de todas as demonstrações em* T. *Para distinguir as descrições temos de codificar* T *numa versão auto-delimitada, isto é, em no máximo* $2k$ *bits. Então para uma constante independente de* T *e de* c *temos* $K(x) \leq 2k + \log c + c'$, *o que contradiz* $K(x) \geq c$ *para todos os* $c > c_T$, *com* $c_T = 3k + c''$.

Enumerabilidade de conjuntos e complexidade de Kolmogorov

Usando a complexidade de Kolmogorov podemos quantificar a distinção entre conjuntos r.e. e recursivos.

Definição 4.0.1 *A* sequência característica *dum conjunto* $A \subseteq \mathbb{N}$ *é uma sequência binária infinita* $\chi = \chi_1\chi_2\ldots$ *definida por*

$$\chi_i = \begin{cases} 1 & \text{se } i \in A \\ 0 & \text{caso contrário} \end{cases}$$

Se A e $\overline{A}$ são r.e então a função $f(i) = \chi_i$ é recursiva e a complexidade condicional $K(\chi_{1:n}|n)$ é limitada por uma constante fixa para todo o n, onde $\chi_{1:n}$ representa a subsequência entre χ_1 e χ_n. Contudo, no caso geral, $K(\chi_{1:n}|n)$ pode crescer ilimitadamente com n, embora logaritmicamente. Isto é, estas sequências são "pouco" aleatórias.

Teorema 4.0.8 *(Lema de Barzdin)*

(i) Qualquer sequência característica χ *de um conjunto r.e.* A *satisfaz* $K(\chi_{1:n}|n) \leq \log n + c$ *para todo o* n, *onde* c *é uma constante que depende de* A *(mas não de* n*).*

(ii) Mais ainda, existe um conjunto r.e. tal que a sua sequência característica χ *satisfaz* $K(\chi_{1:n}) \geq \log n$, *para todo o* n.

Dem. (i) Como A é r.e existe uma função parcial recursiva ϕ tal que A é o domínio de definição de ϕ, isto é, $A = \{x : \phi(x)\downarrow\}$. Usando o método de "dovetail" podemos calcular $\phi(1)$, $\phi(2), \ldots$. Desde modo temos uma enumeração de A pela ordem em que as computações do $\phi(i)$-ésimo termina. O prefixo $\chi_{1:n}$ pode ser reconstruído a partir do número m de 1's que contém. Conhecendo m, podemos usar ϕ para enumerar os elementos de A até ter encontrado m elementos menores ou iguais a n. Seja $B = \{a_1, a_2, \ldots, a_m\}$ o

conjunto desses elementos. Então, de B podemos reconstruir todos os 1's de $\chi_{1:n}$ e os restantes elementos de $\chi_{1:n}$ são 0's. Isto é, podemos descrever $\chi_{1:n}$ a partir duma descrição de ϕ e de m. Sendo $m \leq n$, vem $K(m) \leq \log n + c$ e como $K(\phi) < \infty$ temos o resultado.

(ii) Seja U a máquina de referência universal. Defina-se $\chi = \chi_1\chi_2\ldots$ por

$$\chi_i = \begin{cases} 1 & \text{se } U(\langle i,i\rangle) = 0 \\ 0 & \text{se } U(\langle i,i\rangle) \neq 0 \text{ ou } U(\langle i,i\rangle) \uparrow \end{cases}$$

interpretando $U(\langle i,i\rangle)$ como o cálculo de $\{i\}(i)$. Então, χ é a sequência característica dum subconjunto de $\mathbb{N}$ r.e. (Qual?). Vamos provar que χ verifica a propriedade enunciada. Suponhamos que $K(\chi_{1:n}) < \log n$ para algum n. Seja p o menor programa que calcula $\chi_{1:n}$ com tamanho menor que $\log n$. Logo, $p < n$. Então $U(\langle p,p\rangle) = \chi_p$ o que contradiz a definição de χ_p.

□

Capítulo 5

Informação Algorítmica

Se a complexidade condicional $K(x|y)$ é muito menor que $K(x)$, isso significa que y contém informação sobre x

Definição 5.0.2 *A* informação algorítmica *de x contida em y é:*

$$I(y:x) = K(x) - K(x|y)$$

Considerando a máquina de referência U tal que $U(\langle \epsilon, x \rangle) = x$ então

$$K(x|x) = 0$$

e

$$I(x:x) = K(x)$$

Isto é, dado que esta igualdades são válidas a menos uma constante aditiva, independente de x, $K(x)$ pode ser vista como informação (algorítmica) contida no próprio x. Como já se disse, esta definição da quantidade de informação tem a vantagem de se referir a objectos individuais, e não a objectos considerados como elementos dum conjunto de objectos com uma distribuição de probabilidade (Teoria da Informação de Shannon).

Contudo, ao contrário do que sucede na Teoria da Informação clássica, as igualdades $I(x:y) = I(y:x)$ e $K(x,y) = K(x) + K(y|x)$ só se verificam a menos um factor logarítmico.

Teorema 5.0.9 *(Komolgorov) Para todo os $x,\ y \in \mathbb{N}$,*

$$K(x,y) = K(x) + K(y) + O(min\{\log K(x), \log K(y|x)\})$$

Como $K(x,y) = K(y,x)$ a menos uma constante aditiva tem-se

Corolário 5.0.2 *A menos um termo aditivo* $O(min\{\log K(x), \log K(y)\})$,

$$K(x) - K(x|y) = K(y) - K(y|x)$$

e portanto,

$$|I(x:y) - I(y:x)| = O(min\{\log K(x), \log K(y)\})$$

Exemplo 14 *Para cada* n *existe uma sequência* x *de tamanho* n *tal que* $K(x|y) \leq n$. *Analogamente existe uma infinidade de valores* n *tal que* $K(n) \geq |n|$. *Escolhendo* x *tal que o seu tamanho* n *é aleatório (naquele sentido), temos a menos constantes aditivas:*

$$\begin{aligned} I(x:n) &= K(n) - K(n|x) \geq |n| \\ I(n:x) &= K(x) - K(x|n) \leq n - n = 0 \end{aligned}$$

Isto mostra que $|I(x:y) - I(y:x)|$ *pode ser da ordem do logaritmo das complexidades de* x *e de* y.

Capítulo 6

Complexidade de Kolmogorov auto-delimitada

A complexidade algorítmica definida na secção anterior apresenta alguns problemas tanto do ponto de vista teórico como prático, o que levou ao seu refinamento (10 anos depois . . . e em particular por Chaitin [Cha87]) designado por complexidade de Kolmogorov auto-delimitada, K'. No entanto, como as duas diferem de um factor logarítmico, para muitas aplicações é indiferente qual a versão a usar.

Definição 6.0.3 *Seja Σ um alfabeto. Uma linguagem $A \subseteq \Sigma^\star$ é* independente de prefixos *se nenhum elemento de A é prefixo de outro elemento de A. Uma codificação $c : \Sigma^\star \longrightarrow \mathbb{N}$ é uma* codificação prefixa *se o seu domínio é independente de prefixos.*

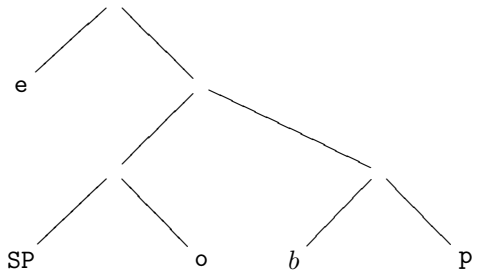
Exemplo 15 *O conjunto $\{0^i 1 \ : \ i = 0, 1, 2, \ldots\}$ é independente de prefixos. O único conjunto independente de prefixos que contém ϵ é $\{\epsilon\}$.*

Exemplo 16 *Seja $c(0) = 1$, $c(10) = 2$, $c(110) = 3$ e $c(111) = 4$. Então, o código 0111100 pode ser descodificado como 1421 duma maneira única!*

Exemplo 17 *O conjunto dos números de telefone (de alfabeto $\{0, 1, \cdots, 9\}$) é independente de prefixos. Não podem existir, por exemplo, telefones "5502631" e "550263".*

Exemplo 18 *As linguagens independentes de prefixos estão na base de um método de compressão de ficheiros que corresponde ao chamado código de Huffman. A cada caracter corresponde uma sequência de $\{0, 1\}^\star$, de modo a que aos caracteres mais frequentes no ficheiro correspondam sequências mais curtas. O algoritmo de Huffman examina o ficheiro a comprimir determinando a*

frequência relativa (número de ocorrências/número total de caracteres) de cada caracter existente. Em seguida constrói uma árvore binária óptima – correspondente a um código que permite uma compressão máxima (utilizando este tipo de códigos!) do ficheiro. Por exemplo, se os caracteres que ocorrem forem $\{\texttt{SP}, \texttt{b}, \texttt{e}, \texttt{o}, \texttt{p}\}$ *(*SP *representa o caracter "espaço") e se as correspondentes frequências forem 0.15, 0.15, 0.4, 0.14, 0.16 uma árvore binária óptima é*



O código de um caracter é obtido a partir do caminho desde a raiz até ele, tomando-se um 0 cada vez que se segue pela sub-árvore esquerda e um 1 quando se segue pela sub-árvore direita. Neste exemplo, temos: $h(\texttt{SP}) = 100$, $h(\texttt{b}) = 110$, $h(\texttt{e}) = 0$, $h(\texttt{o}) = 101$, *e* $h(\texttt{p}) = 111$. *Note-se que o conjunto* $\{0, 100, 101, 110, 111\}$ *é independente de prefixos.*
Que representa a sequência 11101110100110101110101*?*

Como é fácil de se verificar, uma vantagem destas linguagens é que uma qualquer concatenação de palavras pode ser descodificada de uma forma única nas palavras componentes: quando se concatenam palavras não se perde informação.

Uma codificação prefixa de $x \in \mathbb{N}$ é por exemplo 1^x0. Outros exemplos são as versões auto-delimitadas de x como $\overline{|x|}x$. A noção de auto-delimitação pode ser generalizada. Para $x \in \{0,1\}^\star \setminus \{\epsilon\}$, seja $E_0(x) = \overline{x}$ a versão auto-delimitada de ordem 0; $E_1(x) = \overline{|x|}x = x'$ a versão auto-delimitada de ordem 1 ("standard"); $E_i(x) = E_{i-1}(|x|)x$ para $i \geq 1$.

Exercício 6.0.7 Dado x, a versão auto-delimitada de ordem 0 pode ser $E_0(x) = 0^x1$. Obtenha $E_i(x)$ neste caso e determine um majorante das suas complexidades.

Uma das propriedades importantes das codificações (códigos) prefixas é a seguinte, denominada *Inequação de Kraft*:

Teorema 6.0.10 *Para qualquer sequência finita ou infinita de número naturais* l_1, l_2, ..., *existe uma codificação prefixa* c *tal que esta sequência corresponde aos tamanhos das sequências no domínio de* c *se e só se*

$$\sum_n 2^{-l_n} \leq 1$$

Dem. ($\Rightarrow$) Considere-se a injecção entre sequências binárias x e os intervalos reais, $\Gamma_x = [0.x, 0.x + 2^{-|x|}[$ em $[0,1[$. O comprimento de cada intervalo é $2^{-|x|}$. Basta ver que uma codificação prefixa corresponde a um conjunto de intervalos daquela forma disjuntos. Isto prova que a inequação se verifica para codificações prefixas.

($\Leftarrow$) Suponhamos que $l_1, l_2, \ldots$ verificam a inequação. Podemos supor que a sequência é não decrescente. Escolham-se intervalos disjuntos adjacentes I_1, I_2, ... de comprimentos 2^{-l_1}, 2^{-l_2}, ..., a partir do extremo inferior de $[0,1[$. Deste modo, para cada $n \geq 1$ o extremo inferior de I_n é $\sum_{i=1}^{n} 2^{-l_i}$ e o extremo superior de I_n coincide com o extremo inferior de I_{n+1}. Dado que os l_i são não decrescentes, cada intervalo $I_n = \Gamma_x$ para alguma sequência binária x com tamanho $|x| = l_n$. Tome-se $c(x) = n$ para cada I_n. $\square$

Exercício 6.0.8 Construa uma codificação prefixa associada à sequência $2, 3, 6, 10$ e verifique a Inequação de Kraft.

Exemplo 19 *O Teorema 6.0.10 pode-se enunciar em termos de linguagens independentes de prefixos. Para cada $L \subseteq \{0,1\}^\star$ define-se $FI(L) = \sum_{w \in L} 2^{-|w|}$, onde para cada $w \in L$, $2^{-|w|}$ é o indicador de frequência (ou a probabilidade) de w. Se L é independente de prefixos então $FI(L) \leq 1$.*

Por exemplo, para $\{0^i 1 \ : \ i = 0, 1, 2, \ldots\}$ temos

$$FI(L) = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$$

Voltemos às complexidades ...

A complexidade de Kolmogorov auto-delimitada requere que os programas sejam codificações prefixas. Isto é, as máquinas de Turing usadas e nomeadamente a máquina de Turing de referência U, só aceitam programas p tal que nenhum prefixo de p é um programa aceite por essas máquinas. Isto é, o conjunto de programas para os quais estas máquinas param é livre de prefixos. Formalmente,

Definição 6.0.4 *Uma função $\phi : \Sigma^\star \longrightarrow \mathbb{N}$ parcial recursiva é* prefixa *se $\phi(x) \downarrow$ se e só se $\phi(y) \uparrow$ para todo o y prefixo próprio de x.*

Exemplo 20 *Pode-se obter uma enumeração das funções recursivas prefixas usando o algoritmo a seguir apresentado. Considere-se uma enumeração de máquinas de Turing T_1, T_2,.... Vamos considerar uma enumeração apenas das máquinas de Turing que calculam funções prefixas. Seja T uma máquina de Turing tal que os seus dados podem ser apenas lidos da esquerda para a direita sem nunca voltar a trás. Suponhamos que $x = b_1 b_2 \ldots$ são os seus dados. Então, modificámos T para executar o seguinte algoritmo:*

Passo 1: *Seja* $x := \epsilon$

Passo 2 *Consideram-se as computações de* T *pelo método de "Dovetail" de* $\phi(xy)$ *para todo o* $y \in \Sigma^\star$. *Se* $\phi(xy) \downarrow$ *e é a primeira computação que pára então vá para* **Passo 3.**

Passo 3: *Se* $y = \epsilon$ *então produza* $\phi(x)$ *e pare; senão leia o bit seguinte de* x; $x := xb$; *vá para* **Passo 2**.

Esta construção produz uma enumeração das máquinas $T'_1, \ldots$ *que calculam funções recursivas prefixas. (Verifique!)*

Exemplo 21 *Uma forma alternativa de definir uma máquina de Turing* prefixa *é a seguinte: Considere-se a classe de máquinas de Turing* PM *com uma fita semi-infinita de leitura, uma fita semi-infinita de escrita e uma fita duplamente infinita de "trabalho". Suponhamos que a fita de leitura apenas contém* 0*'s e* 1*'s (não contém brancos). No estado inicial os dados são escritos na fita de leitura, e a cabeça de leitura encontra-se na primeira célula. Inicialmente as restantes fitas apenas contêm brancos. Cada passo da computação — transição da máquina de Turing consiste numa de três acções:*

- *A máquina lê o símbolo seguinte da fita de dados movendo a cabeça da fita de dados de uma unidade para a direita e alterando o seu estado de acordo com o símbolo lido.*
- *A máquina escreve um símbolo (dependente do seu estado) na fita de resultados, movendo a cabeça da fita de resultados de uma unidade para a direita.*
- *A máquina executa uma transição relativa à fita de trabalho semelhante às transições nas máquinas de uma só fita.*

Quando a máquina parar o resultado x *é o conteúdo da fita de escrita. Denominamos esta máquinas de Turing auto-delimitadas, porque o conjunto de dados para as quais cada máquina pára é independente de prefixos. Uma sequência* p *é um programa para* T *se* T *pára com a cabeça de leitura no bit mais à direita de* p *(isto é,* p *é um prefixo maximal duma sequência infinita de dados). Qualquer programa da forma* px *é exactamente equivalente ao programa* p.

Exercício 6.0.9 Mostrar que a classe de máquinas de Turing auto-delimitadas PM, definidas no exemplo anterior, calculam precisamente o conjunto de funções parciais recursivas prefixas.

O Teorema da Invariância continua válido se nos restringirmos a este tipo de máquinas.

Exercício 6.0.10 Mostre a afirmação anterior.

Podemos então fixar uma máquina de Turing U' auto-delimitada de referência. A *complexidade auto-delimitada (prefixa)* de x é definida por $K'(x) = K_{U'}(x)$ isto é o tamanho do menor programa p de U' tal que $U'(p) = x$. Analogamente se definem complexidades auto-delimitadas condicionais.

Sempre que for explícito pelo contexto, qual a complexidade ou qual a máquina de referência a que nos referimos, designaremos estes valores apenas por K ou U.

A complexidade auto-delimitada K' verifica várias propriedades sem o factor logarítmico que afectava os resultados para K.

Exemplo 22 *Seja $K'(x,y) = K'(\langle x,y\rangle)$. Neste caso, como sabemos que os programas são auto-delimitados (as codificações são prefixas), podemos concatenar descrições sem ter de marcar as suas extremidades. Então,*

$$K'(x,y) \leq K'(x) + K'(y) + O(1)$$

Nomeadamente, seja U' a máquina de referência. Sejam $U'(x^\star) = x$ com $K'(x) = |x^\star|$ e $U'(y^\star) = y$ com $K'(y) = |y^\star|$. Seja V uma máquina universal auto-delimitada que primeiro lê $x^\star$ e calcula x, depois lê $y^\star$ e calcula y e finalmente calcula $\langle x,y\rangle$. Seja $n = e(V)$ para uma enumeração de máquinas de Turing auto-delimitadas. Então U' com dados $0^n 1 x' y'$ calcula $\langle x,y\rangle$.

Exemplo 23 *As funções K e K' são assimptoticamente iguais. Para todo o x, y tem-se a menos duma constante aditiva*

$$K(x|y) \leq K'(x|y) \leq K(x|y) + 2\log K(x|y)$$

A primeira inequação é trivial (Porquê?). Seja $K(x|y) = |p|$ tal que U calcula x com dados $\langle y,p\rangle$. A codificação $p' = \overline{|p|}p$ é uma codificação auto-delimitada de p. Por isso,

$$K'(x|y) \leq K(x|y) + 2|K(x|y)| + O(1)$$

Em consequência do Teorema 3.0.3 podemos concluir que para todo o x, $|x| = n$

$$K'(x) \leq n + 2\log n + O(1)$$

embora usando outras codificações prefixas este majorante possa ser melhorado.

O Teorema da Incompressibilidade pode-se enunciar do seguinte modo:

Teorema 6.0.11 *(i)* $\forall n, max\{K'(x) \ : \ |x| = n\} = n + K'(n) + O(1)$

(ii) $\forall c, \#\{x \ : \ |x| = n \ e \ K'(x) \leq n + K'(n) - c\} \leq 2^{n-c+O(1)}$

Diz-se que $x \in \Sigma^\star$ com $|x| = n$ é incompressível se $K'(x) \geq n$.

Exercício 6.0.11 Relacione as noções de incompressibilidade para K e K'.

Teorema 6.0.12 *A menos duma constante aditiva tem-se*

$$K'(x,y) = K'(x) + K'(y|(x,K'(x)))$$

e

$$K'(x,K'(x)) = K'(x)$$

Então ,

Teorema 6.0.13 *A menos duma constante aditiva tem-se*

$$K'(y) - K(y|(x,K'(x))) = K'(x) - K(x|(y,K'(y)))$$

isto é,

$$I'(\langle x,K'(x)\rangle : y) = I'(\langle y,K'(y) >: x) + O(1)$$

O problema da paragem resolvido a partir de $K'(x)$

Vamos ver que o conhecimento de $K'(x)$ permitiria resolver o problema da paragem ("halting problem"); uma consequência deste facto é que a função $K'(x)$ não é computável. Comecemos pelo seguinte Lema.

Lema 6.0.2 *Seja p um programa com n bits.*

a) Se o programa pára, o tempo t que demora a parar satisfaz (onde c é uma constante)

$$K'(t) \leq n + c$$

b) Como consequência, se $t \geq T$ implica $K'(t) > n + c$, e se o programa já correu o tempo T sem parar, então o programa nunca vai parar.

Dem. a) Consideremos uma máquina de Turing M, semelhante à máquina universal U, mas com as seguintes diferenças: M conta o número t de passos que executa e no fim (se parar) dá t como resultado. A simulação de M em U permite calcular t com $n+c$ bits.

b) Se o programa parasse, tinhamos uma descrição de $t \geq T$ com $n+c$ bits o que contraria $K'(t) > n + c$.

□

Seja n um inteiro e consideremos uma palavra x_{max} de comprimento n com complexidade K' máxima (isto é, se $|x| = n$, então $K'(x) \leq K'(x_{max})$). Sabemos do Teorema 6.0.11 que

$$K'(x_{max}) = n + K'(n) + O(1)$$

donde

$$K'(x_{max}) > n + K'(n) - c'$$

para uma constante apropriada c'.

Vamos agora procurar um majorante de $K'(x_{max})$. Seja n um inteiro fixo. O conjunto dos majorantes de $K'(x)$ com $|x| = n$ é recursivamente enumerável. A prova deste facto pode basear-se na utilização da técnica do "dovetailing" para a máquina de referência U e para todos os x de comprimento n e programas possíveis (λ, 0, 1, 00,...). Quando (e se) a máquina simulada parar para um certo programa p com resultado x então temos um majorante, $K'(x) \leq |p|$.

Seja então dado o inteiro n e consideremos a enumeração referida. Se $K'(x)$ fosse computável, seria possível terminar a computação quando todos os valores de $K'(x)$ para x de comprimento n (em particular para x_{max}) tivessem sido atingidos. Seja $\beta(n)$ o correspondente tempo. A partir de n e de $\beta(n)$ ou de qualquer valor $t \geq \beta(n)$ seria possível determinar x_{max}. Teríamos (recordemos que estamos a utilizar complexidades auto-delimitadas)

$$K'(x_{max}) \leq K'(t) + K'(n) + c''$$

Obtemos então

$$n + K'(n) - c' < K'(x_{max}) \leq K'(t) + K'(n) + c''$$

donde

$$K'(t) > n - c' - c''$$

Concluímos que, se $t \geq \beta(n)$ então $K'(t) > n - c' - c''$. Poderíamos então utilizar $\beta(n)$ (que, como vimos, seria computável se K' o fosse) e o Lema anterior para decidir da seguinte forma o problema da paragem: um programa p com n bits pára sse pára antes do tempo $\beta(n)$.

Capítulo 7

Aplicações da Compressibilidade

Sendo ω uma sequência infinita de 0's e 1's denota-se por $\omega_{1:n}$ o seu prefixo de tamanho n. Diz-se que ω é *aleatória* se

$$\exists c \, \forall n K'(\omega_{1:n}) \geq n - c$$

Um número real $r \in \mathbb{R}$ pode ser visto como a sequência, possivelmente infinita, dos seus dígitos (numa dada base). Dizemos que um número é *computável* se existe um algoritmo que permite calcular a sua sequência de dígitos. Formalmente uma sequência infinita ω é *recursiva* sse existe uma função recursiva $f : \mathbb{N} \longrightarrow \Sigma^\star$ tal que $\omega_{1:n} = f(n)$. Pode-se mostrar que ω é recursiva sse $K(\omega_{1:n}) \leq K(n) + c$. Neste sentido podemos concluir que não existem números reais computáveis que sejam aleatórios (isto é que a sua sequência de dígitos seja aleatória).

Isto aplica-se por exemplo a números como e ou π. Em conclusão, não devíamos usar a sequência de dígitos de π como base dum gerador de números aleatórios. Uma outra caracterização de números aleatórios é o de serem *normais*, isto é, se, para cada $n \geq 1$, qualquer sequência de dígitos de 0 a 9 (se a base for 10) ocorrer com a mesma probabilidade assimptótica; por exemplo, a probabilidade da sequência "87" é, no limite, 1/100. Nenhum número racional é *normal* e números como o e ou o π não se sabe. Contudo, existem números normais (na base 10) que não são "nada"aleatórios. Um desses números é o número de Champernowne:

$$0.12345678910111213114 15\ldots$$

Contudo este número não é normal, por exemplo na base 2.

Por outro lado, quase todos os números reais são aleatórios (no sentido de Kolmogorov) (Porquê?). Mas como obter um tal número?

Considere-se o seguinte número não computável. Seja U uma máquina universal de Turing e $T_1, T_2, \ldots$ uma enumeração de máquinas de Turing. Define-se $k = 0.k_1k_2\ldots$ tal que $k_i = 1$ se T_i

pára com a fita vazia e $k_i = 0$ caso contrário. Pela insolubilidade do problema da paragem k não é computável. Vamos ver que também não é aleatório. O conjunto $B = \{i \ : \ U(T_i) \downarrow\}$ é r.e., logo, pelo Lema de Barzdin, k é compressível: cada prefixo $k_{1:n}$ de tamanho n, para todo $n \geq n_0$, pode ser calculado por uma sequência de tamanho no máximo $2 \log n$ (Verifique porquê!).

O número Ω

Considerando a máquina universal de referência, para cada $x \in \Sigma^\star$ podemos definir a sua *probabilidade* por $m(x) = \sum_{U'(p)=x} 2^{-|p|}$

A função $m(x)$ pode ser interpretada como a probabilidade de U' parar e produzir x, tendo como dados um programa p obtido por $|p|$ lançamentos de uma moeda ao ar.

Lema 7.0.3 *Para todo o* $x \in \Sigma^\star$

(i) $m(x) \geq 2^{-K'(x)}$

(ii) $0 < m(x) < 1$

(iii) $0 < \sum_{x \in \Sigma^\star} m(x) < 1$

Dem. A alínea (i) verifica-se porque $2^{-K'(x)}$ é um dos termos de $m(x)$. Pela inequação de Kraft e porque $L = \{p \ : \ U'(p) = x\}$ é uma linguagem livre de prefixos, tem-se que $m(x) \leq 1$. O mesmo argumento se aplica para o limite superior de (i) Como todos os termos dos somatórios são não negativos tem-se $m(x) \geq 0$ e analogamente para (iii). A primeira inequação de (ii) é estrita por (i) e a segunda inequação é estrita porque $\sum_{x \in \Sigma^\star} m(x) \leq 1$. A segunda inequação de (iii) é estrita porque existem programas para os quais U' não pára. □

Em particular pode-se provar que $K'(x) = -\log m(x) + O(1)$ (Solomonoff-Levin).

O número Ω denominado *probabilidade da paragem* da máquina de referência U' (ou "número mágico") é definido por:

$$\Omega = \sum_{U'(p)\downarrow} 2^{-|p|} = \sum_x m(x)$$

Pelo Lema 7.0.3:

$$0 < \Omega < 1$$

Seja

$$\Omega = 0.b_1 b_2 \ldots$$

a representação de Ω na base[1] 2. Denote-se por $B = b_i b_2 \ldots$ a sequência (possivelmente infinita ...) dos seus bits e $B_{1:n} = b_1 \ldots b_n$ o seu prefixo de tamanho n, para $n \geq 1$. Para todo $n \geq 1$ define-se o número racional

$$\Omega_n = 0.b_1 \ldots b_n$$

O número Ω tem as seguintes propriedades e as seguintes implicações para sistemas formais:

- Ω codifica o problema da paragem.
- B é aleatória.
- Qualquer sistema formal axiomático apenas pode produzir um número finito de bits de Ω.
- Para todo o sistema formal axiomático T que possa ser expresso em n bits, $B_{1:n}$ pode ser usada para decidir se uma qualquer fórmula bem formada, em T é um teorema, um não teorema ou independente.
- Um sistema formal só pode deduzir $B_{1:n}$ se $B_{1:n}$ for dado como axioma.
- Em conclusão, podemos definir Ω formalmente mas não existe nenhum algoritmo que permita calcular uma infinidade de bits de Ω.
- Existe uma equação diofantina $A(n, x_1, \ldots, x_m) = 0$ que tem um número infinito de soluções se e só se o n-ésimo bit de Ω é 1.
- Podemos "codificar" Ω na teoria elementar dos números.
- A aleatoriedade é inerente à Matemática.

O conjunto $D = \{p \ : \ U'(p) \downarrow\}$ é r.e.. Consideremos uma enumeração $g : \mathbb{N} \longrightarrow \Sigma^\star$ de D obtida pelo método de "dovetail" das possíveis computações. Recapitulando, para $i = 1, \ldots$ a fase i consiste em U' executar o j-ésimo passo de computação nos k-ésimos dados, para todos os j e k tal que $j + k = i$. Define-se $g(1)$ como o primeiro programa para o qual a computação de U' pára, $g(2)$ o segundo, ...

Para todo o $i \geq 1$ seja

$$\omega_i = \sum_{j=1}^{i} 2^{-|g(j)|}$$

É óbvio que a sucessão ω_i é monótona crescente e $\lim_{i\to\infty} \omega_i = \Omega$

Lema 7.0.4 *Se $\omega_i > \Omega_n$ então*

$$\Omega_n < \omega_i < \Omega \leq \Omega + 2^{-n}$$

[1] Se Ω for racional considera-se uma representação com um número infinito de zeros

Dem. A última inequação resulta de que

$$2^{-i} \geq \sum_{j=i+1}^{\infty} b_j 2^{-j}$$

para quaisquer $b_j \in \{0, 1\}$. □

Teorema 7.0.14 *Dado um qualquer n, se se conhecer $B_{1:n}$ então é possível decidir o problema da paragem para qualquer programa p com tamanho $|p| \leq n$.*

Dem. Conhecendo $B_{1:n}$ pode-se construir Ω_n. Determina-se i tal que ω_i verifica $\omega_j > \Omega_n$. Pelas propriedades da sucessão ω_i, tal i existe sempre. Seja p_1 um programa tal que $|p_1| = n_1 \leq n$. Então $U'(p_1) \downarrow$ se e só se p_1 é um dos $g(1), \ldots, g(i)$. Que a condição é necessária é trivial. Suponhamos que $U'(p_1) \downarrow$ mas que $p_1 = g(m)$ e $m > i$. Então

$$\Omega > \omega_m \geq \omega_i + 2^{-n_1} \geq \omega_i + 2^{-n} > \Omega_n + 2^{-n} \geq \Omega$$

Absurdo! □

Pelo Teorema 7.0.14, o conhecimento dum prefixo suficientemente grande $B_{1:n}$ permite resolver qualquer problema da paragem e n pode ser determinado pelo . Em particular, permitiria resolver qualquer problema que possa ser refutado por contra-exemplos finitos, o que é o caso de muitas das conjecturas em matemática.

Exercício 7.0.12 O Teorema 7.0.14 contradiz a indecidibilidade do problema da paragem? Justifica a resposta. Porque é que Ω é não computável?

Teorema 7.0.15 *A sequência B é aleatória.*

Dem. Recorde que B é aleatória se

$$\exists c \forall n \; K'(B_{1:n}) \geq n - c$$

Dado $B_{1:n}$ é possível calcular todos os programas p de comprimento não superior a n para os quais U' converge. Então,

$$\{U'(g(j)) \; : \; 1 \leq j \leq i \text{ e } |g(j)| \leq n\} = \{x \; : \; K'(x) \leq n\}$$

Se x não pertence a este conjunto, tem-se necessariamente que $K'(x) > n$. Então é possível construir uma função recursiva ϕ tal que $\phi(B_{1:n}) = x$. Vem

$$K'(\phi(B_{1:n})) \leq n$$

Dada uma descrição de ϕ em c bits, para cada n pode-se calcular $\phi(B_{1:n})$ de $B_{1:n}$, o que significa que:

$$K'(B_{1:n}) + c \leq n$$

□

Exercício 7.0.13 Diga como se pode construir a função ϕ referida na demonstração anterior. Sugestão: Sendo $x = x_1 \ldots x_t$ considere m o menor inteiro, se existir algum, tal que $\omega_m = \sum_{j=1}^{t} x_j 2^{-j}$. Se tal m existir considere $f(x)$ a primeira sequência que não pertence a $\{g(j) : 1 \leq j \leq m\}$.

Teorema 7.0.16 *(Chaitin,[Cha87]) Existe uma equação diofantina $A(n, x_1, \ldots, x_m) = 0$ que tem um número infinito de soluções $x_1, \ldots, x_m$ se e só se o n-ésimo bit de Ω é 1.*

Chaitin [Cha87, Cha94] construiu explicitamente a equação mencionada no teorema anterior, com cerca de 17000 variáveis.

Exercício 7.0.14 Justifica as implicações da existência do número Ω acima referidas.

Capítulo 8

Minorantes de Complexidade

A existência de sequências incompressíveis, aleatórias segundo Kolmogorov, permite estabelecer minorantes da complexidade de vários problemas e algoritmos. As demonstrações por contagem para estabelecer minorantes, envolvem todas as instâncias (sequências dum dado tamanho) duma classe de problemas e prova-se que o minorante deve verificar-se para algumas dessas instâncias.

Numa demonstração típica, usando o método da incompressibilidade, escolhe-se uma sequência aleatória duma dada classe (que existe, mas que não se pode exibir...). Essa sequência é incompressível. Depois prova-se que a propriedade desejada (por exemplo, um minorante de complexidade) é verificada por essa sequência. O argumento é que se a propriedade não se verificasse, então essa sequência era compressível.

Dado que apenas se tem de manipular um objecto, as demonstrações são normalmente mais simples e curtas.

Vamos apenas considerar três exemplos.

8.0.1 Máquina de Turing com uma fita

Uma máquina de Turing M com uma fita semi-infinita necessita de $\Omega(n^2)$ passos para reconhecer $L = \{xx^R \ : \ x \in \Sigma^\star\}$, isto é, a sua complexidade temporal $T_M(n)$ é $\Omega(n^2)$.

Já vimos que existe uma máquina de Turing M que reconheça L em $O(n^2)$ passos. Vamos ver que qualquer máquina de Turing deste tipo necessita $\Omega(n^2)$ passos para reconhecer L.

Durante uma computação de M podemos associar a cada célula da fita[1] c_i uma *sequência de passagem* $sp_i = (q_1, \ldots, q_m)$ constituída pelos estados de M nos passos em que a cabeça passa dessa célula para a célula à sua direita, primeiro da esquerda para a direita, e depois alternadamente em ambas as direcções. Denota-se por $|sp_i|$ o tamanho duma sequência de passagem para a célula c_i.

[1] que contém inicialmente os dados

A demonstração é feita por redução ao absurdo. Suponhamos que $T_M(n)$ é $o(n^2)$. Suponhamos que M pára na primeira célula depois dos dados (marcador). Seja $|M|$ o tamanho duma descrição de M. Fixemos x uma sequência de tamanho $|x| = n$ tal que $K(x|(M,n)) \geq n$. Pelo Teorema da Incompressibilidade x existe. Consideremos uma computação de M com dados $x0^{2n}x^R$. Se cada sequência de passagem associada a uma célula no segmento 0^{2n} é maior que $n/(2|M|)$, então M executa pelo menos $n^2/|M|$ passos. Caso contrário, existiria uma sequência de passagem sp_i de tamanho menor que $n/(2|M|)$, associada a uma célula c_i. Esta sequência de passagem sp_i é completamente descrita por no máximo $n/2$ bits[2]. Usando sp_i, é possível reconstruir x por verificação exaustiva de todas as sequências binárias de tamanho n. Para cada y sequência candidata de tamanho n, coloca-se a sequência $y0^{2n}$ nas primeiras $3n$ células mais à esquerda da fita e simula-se a computação de M desde o estado inicial. Cada vez que a cabeça passa da célula c_i para a célula à sua direita, interrompe-se a computação de M com a cabeça à direita de c_i e recomeça-se a computação q obtido de sp_i e com a cabeça da fita na célula c_i. Assim M não executa "realmente" nenhum movimento à direita de c_i

Suponhamos que nesta computação com y, cada vez que a cabeça se movimenta de c_i para a sua direita, o estado corrente de M é o estado seguinte indicado por sp_i. Então M aceita os dados $y0^{2n}x^R$. Isto é, a computação à direita da célula c_0 é idêntica à computação correspondente com dados $x0^{2n}x^R$. Dado que M pára à direita de c_0, ou aceita ambas as sequências $y0^{2n}x^R$ e $x0^{2n}x^R$ ou rejeita as duas. Como M aceita L tem-se que $y = x$. Assim, dado sp_i podemos reconstruir x com um programa cujos dados são a informação de M, n e sp_i. Isto significa que

$$K(x|(M,n)) \leq |sp_i| + O(1) \leq n/2 + O(1)$$

o que contradiz $K(x|(M,n) \geq n$, para n grande.

8.0.2 Linguagens Regulares

Recorde que uma linguagem é regular se e só se for aceite por um autómato finito, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ onde $\delta : \Sigma \times Q \longrightarrow Q$ é a função de transição e $F \subseteq Q$ são os estados finais. A linguagem aceite por $\mathcal{A}$ é $L = \{x \ : \ \delta'(x, q_0) \in F\}$, onde δ' é δ estendida a $\Sigma\star$ por $\delta'(q, ax) = \delta'(q', x)$ se $\delta(q, a) = q'$ e $\delta'(q, \epsilon) = q$. Usando a complexidade de Kolmogorov vamos mostrar que $L = \{0^k 1^k \ : \ k \geq 1\}$ não é regular. Suponhamos que era regular. Fixe-se k com $K(k) \geq \log k$. Seja q o estado dum autómato $\mathcal{A}$ que aceite L, depois de processar 0^k. Então q e A são uma descrição de k. Nomeadamente, executando $\mathcal{A}$ iniciando em q e com uma sequência de 1's, $\mathcal{A}$ chega a um estado de aceitação exactamente após k uns consecutivos. A descrição de $\mathcal{A}$ e q estão limitados por uma

[2]Note que cada estado é descrito com menos de $|M|$ bits!

constante, seja c, independente de k. Então, $K(k) \leq c + O(1)$ o que contradiz a hipótese, para k suficientemente grande.

O lema seguinte generaliza este exemplo

Lema 8.0.5 *(Regularidade-KC) Seja $L \subseteq \Sigma^\star$ uma linguagem regular e $L_x = \{y \ : \ xy \in L\}$. Existe uma constante c, tal que para cada x, se y é a n-ésima[3] sequência em L_x, então $K(y) \leq K(n) + c$*

Dem. Seja L regular. A n-ésima sequência y tal que $xy \in L$ pode ser descrita por:

- por pertencer a L_x e pela descrição do autómato finito que descreve L
- o estado do autómato depois de processar x e pelo número n

O primeiro item requere $O(1)$ bits. Então $K(y) \leq K(n) + O(1)$ $\square$

Exercício 8.0.15 Prove que $\{1^p \ : \ p \text{ é primo}\}$ não é regular.
Sugestão: considere $xy = 1^p$ com p o $k+1$-ésimo primo. Seja $x = 1^{p'}$ (do lema) com p' o k-ésimo primo. Então, $y = 1^{p-p'}$ e $n = 1$. Use o facto de que a diferença entre primos consecutivos crescer de forma não limitada.

Exercício 8.0.16 Prove que $\{0^i 1^j \ : \ i \neq j\}$ não é regular.
Sugestão: Considere $x = (0)^m$ e $K(m) \geq \log m$.

Exercício 8.0.17 Prove que $\{xx^R \ : \ x \in \Sigma^\star \setminus \{\epsilon\}\}$ não é regular.
Sugestão: Considere $x = (01)^m$ e $K(m) \geq \log m$. O primeiro elemento de L_x é $y = (10)^m 0$. Qual é $K(y)$?

Usando a complexidade de Kolmogorov pode ainda caracterizar-se completamente as linguagens regulares.

Definição 8.0.5 *Considere-se um enumeração de $\Sigma^\star$ (por exemplo, a lexicográfica) sendo y_i o seu i-ésimo elemento. Seja $L \subseteq \Sigma^\star$ e $x \in \Sigma^\star$, considere-se a sequência característica $\chi = \chi_1 \chi_2 \ldots$ de $L_x = \{y \ : \ xy \in L\}$, tal que $\chi_i = 1$ se $xy_i \in L$, caso contrário $\chi_i = 0$.*

Teorema 8.0.17 *(Caracterização de regularidade (KC)) Seja $L \subseteq \Sigma^\star$. Existe uma constante c_L dependente de L, tal que as seguintes afirmações são equivalentes:*

(i) L é regular

(ii) para todo o $x \in \Sigma^\star$, para todo o n, $K(\chi_{1:n}|n) \leq c_L$

[3] na ordem lexicográfica

(iii) para todo o $x \in \Sigma^\star$*, para todo o* n*,* $K(\chi_{1:n}) \leq K(n) + c_L$

(iv) para todo o $x \in \Sigma^\star$*, para todo o* n*,* $K(\chi_{1:n}) \leq \log n + c_L$

Exercício 8.0.18 *Mostre o Teorema anterior.*
Sugestão: Mostre que $(i) \rightarrow (ii)$ *e* $(ii) \rightarrow (iii) \rightarrow (iv)$*. A parte* $(iv) \rightarrow (i)$ *usa o facto de, para cada* c*, apenas um número finito de sequências infinitas terem para todo o* n*,* $K(\chi_{1:n}) \leq \log n + c$ *(porquê?). Considera-se ainda uma relação invariante à direita* R *tal que* xRx' *se* $\chi = \chi'$ *e usa-se o Teorema de Myhill-Nerode para linguagens regulares.*

8.0.3 Análise da complexidade do caso médio do Heapsort

Seja $A[1] \ldots A[n]$ uma sequência de números inteiros a ordenar por ordem crescente. Uma árvore binária diz-se *essencialmente completa* se cada nó tiver dois filhos, excepto eventualmente um nó no nível 1 que só possui um filho à esquerda. Um `heap` é uma árvore binária essencialmente completa em que para cada nó interno o seu valor é maior ou igual aos valores dos seus filhos. Formalmente, uma sequência de valores $A[1], \ldots, A[n]$ é um `heap` se

$$A[\lfloor j/2 \rfloor] \geq A[j] \text{ para } 1 \leq \lfloor j/2 \rfloor < j \leq n$$

Qualquer sequência A pode ser vista como uma árvore, considerando a raiz $A[1]$ e para o nó $A[i]$ os seus filhos são $A[2i]$ e $A[2i+1]$, com $2i, 2i+1 \leq n$.

Na figura 8.1 apresenta-se um algoritmo de ordenação baseado na construção dum `heap`. A função `makeheap` constrói um `heap` na sequência $a[n]$. Se $a[n]$ for um `heap` no qual um elemento foi modificado (e cujo valor decresceu) a função `siftdown` permite tornar a de novo num `heap`. Esta função executa a seguinte operação: Se $a[i] < a[2i]$ troca-se esse nó pelo filho maior e continua-se esse processo descendo a árvore (`siftdown`). O algoritmo do `heapsort` tem complexidade no pior caso de $O(n \log n)$ sendo a complexidade da função `makeheap` de $O(n)$. Note-se que a profundidade dum `heap` é $\log n$.

Vamos ver que a complexidade no caso médio é também $O(n \log n)$. Suponhamos que as sequências de n valores são igualmente prováveis, isto é, que têm uma distribuição de probabilidade uniforme.

Teorema 8.0.18 *(I. Munro) Em média o* `Heapsort` *faz* $n \log n + O(n)$ *atribuições e* $2n \log n - O(n)$ *comparações.*

Dem. Vamos dar apenas uma ideia da demonstração.

Dados n valores as suas permutações são $n! \approx n^n e^{-n} \sqrt{2\pi n}$. Pelo Teorema da Incompressibilidade existe uma permutação p dos n valores, tal que

$$K(p|n) \geq \log n! - c \geq n \log n - n \log e \geq n \log n - 2n \tag{8.1}$$

```
void swap (int a[],int i,int k){
  int aux;
    aux=a[k];
    a[k]=a[i];
    a[i]=aux;
}
/* ‘‘peneira’’ o nodo i ate a heap estar restabelecida */
void siftdown(int a[],int n, int i){
  int k,j;
  k=i;
  do{
    j=k;
  if(2*j<= n && a[2*j]>a[k])
    k=2*j;
  if(2*j< n && a[2*j+1]>a[k])
    k=2*j+1;
  swap(a,j,k);
  }while (j!=k);
}

void makeheap(int a[],int n){
  int i;
  for(i=n/2;i>=1;i--){
    siftdown(a,n,i);
  }
}

/* para cada i a[1..i] nao esta ordenada e a[i-1...n] esta ordenada por
ordem crescente. A posição a[0] não é usada. */
void heapsort(int a[],int n) {
  int i;
  makeheap(a,n);
/* passo de ordenacao */
  for(i=n;i>=2;i--){
    swap(a,1,i);
    siftdown(a,i-1,1);
  }
}
```

Facto Seja h o `heap` construído por `makeheap` com dados p satisfazendo a equação (8.1). Então,

$$K(h|n) \geq n \log n - 6n \tag{8.2}$$

Dem. Suponhamos por absurdo que $K(h|n) < n \log n - 6n$. Então, vamos mostrar que se pode descrever p, usando h e n, com menos de $n \log n - 2n$ bits. Podemos codificar o processo de construção do `heap` h a partir de p. Em cada ciclo, quando o valor $v = A[i]$ é "empurrado" para baixo na árvore podemos memorizar o caminho que v percorre: 0 indica um ramo esquerdo, 1 um ramo direito e 2 significa que pára. No total este processo pode ser memorizado em $n \log 3 \sum_{i=1} \lfloor n/2 \rfloor i/2^{i+1} \leq 2n \log 3$ bits[4]. Dado o `heap` h final e a descrição dos caminhos, podemos inverter a função `makeheap` e reconstruir p. Então,

$$K(p|n) < K(h|n) + 2n \log 3 + O(1) < n \log n - 2n$$

Obtemos uma contradição. □

Podemos descrever h usando a história dos $n-1$ rearranjos do `heap` durante o passo de ordenação do `heapsort`. Apenas necessitamos de guardar, para $i = n-1, \ldots, 2$ a posição final em que $A[i]$ é inserida. Essa posição pode ser codificada descrevendo o caminho da raiz até essa posição. O caminho pode ser representado por uma sequência binária, com 0 indicando um ramo à esquerda e 1 um ramo à direita. Cada caminho é codificado como um par (l, s), onde $l = |s|$ e l é auto-delimitada e s é a codificação do caminho. Se o i-ésimo caminho tem tamanho d_i, então esta descrição requere no máximo

$$d_i + 2 \log d_i \tag{8.3}$$

Seja H a sequência que resulta da concatenação das descrições de todos os caminhos.

Facto É possível reconstruir h de H e de n.

Dem. Suponhamos que conhecemos H e que h é um `heap` com n valores diferentes. Podemos simular o passo de ordenação em sentido inverso. Inicialmente, $A[1..n]$ contém a sequência ordenada com pelo menos um elemento, $A[1]$.

para $i = 2, \ldots, n-1$ **faça:** Colocar o valor de $A[i]$ em $A[1]$ e deslocar todos os valores no caminho de ordem $n - i$ em H, uma posição para baixo a partir da raiz $A[1]$. O último valor neste caminho não tem para onde ir e é colocado na posição vazia $A[i]$.

[4] Analisando a função `siftdown`

terminação A sequência $A[1],\ldots,A[n]$ é o `heap` h.

□

Tem-se então que $K(h|n) \leq |H| + O(1)$ e pela equação (8.2) vem $|H| \geq n\log n - 6n$. Pela descrição da equação (8.3), isso só é possível se a média dos tamanhos dos caminhos é pelo menos $\log n - c'$, para alguma constante c'. Este valor dá o número médio de atribuições em cada volta do passo de ordenação do `heapsort`. Portanto, começando com um `heap` h, o `heapsort` executa pelo menos $n\log n - O(n)$ atribuições. Dado h, o número de comparações em `siftdown` é $2n\log n - O(n)$.

Dado que a maioria das permutações de n elementos é aleatória, segundo Kolmogorov, estes limites que se verificam para uma sequência aleatória p, também se verificam para todas as permutações em *média*. □

Bibliografia

[Cha87] C. J. Chaitin, *Algorithmic information theory*, Cambridge University Press, 1987.

[Cha94] ———, *Limits of mathematics*, Tech. report, IBM Thomas J. Watson Research Center, 1994.

[LV90] Ming Li and Paul M. B. Vitànyi, *Kolmogorov complexity and its applications*, Handbook of Theoretical Computer Science (J. van Leewen, ed.), vol. A, Elsevier Science Publishers, 1990, pp. 188–254.

[LV94] Ming Li and Paul M. B. Vitàny, *An introduction to kolmogorov complexity and its application*, Texts and Monographs in Computer Science, Springer-Verlag, 1994.

[Sha48] Claude E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), 379–423,623–656, Part I,Part II.