por **Paulo Mateus**
[Instituto Superior Técnico]

O problema P≠NP?

Um milhão de dólares e o nome na eternidade para quem resolver um dos mais importantes problemas matemáticos.

O problema P≠NP é o mais recente dos problemas do milénio listados pelo Clay Mathematics Institute e é, sem sombra de dúvida, o mais importante e influente na área da teoria da computação. A resposta a este problema terá impacto muito significativo em áreas como a criptografia, a optimização, a verificação de modelos, a bioinformática etc., podendo ainda ter consequências importantes na sociedade (por exemplo, o fim do comércio electrónico). A questão foi posta rigorosamente pela primeira vez por Stephen Cook em 1971 [2].

Para entender com algum detalhe o problema P≠NP é essencial compreender o conceito de função computável, bem como o de função computável em tempo polinomial. Neste texto vai-se evitar a abordagem seminal de Turing [3] e das suas máquinas, que à primeira vista é um pouco pesada, considerando-se em alternativa uma arquitectura moderna para o computador[1]. A memória de um computador ξ é uma sucessão de registos $\xi = R_0, R_1, \ldots, R_n, \ldots$ onde em cada registo se pode guardar um natural arbitrariamente grande. O valor guardado no registo R_0 na configuração ξ é denotado por R_0^ξ. Dado que o tamanho de um natural vai ser importante, denotamos por $|R_i|$ o número de *bits* necessários para escrever R_i em base 2, sendo este valor igual a $\lfloor \log_2(R_i)+1 \rfloor$. A escolha da base é irrelevante, mas a base 2 imita a implementação do computador actual. Para inicializar a memória do computador assume-se que os primeiros k registos $(R_0 \ldots R_{k-1})$ tomam os valores $x_0 \ldots x_{k-1}$ e os restantes registos estão a zero. Esta configuração de memória é denotada por $\xi(x_0 \ldots x_{k-1})$.

Para usufruir da memória de um computador é necessário poder alterá-la, o que se faz por intermédio de programas. Apesar de ser costume introduzir uma panóplia variada de programas atómicos, neste texto vamos considerar apenas quatro:

- $R_i := n$ atribui a R_i o natural n;
- $R_i := R_j + R_k$ atribui a R_i a soma de R_j com R_k;
- $R_i := |R_j|$ atribui a R_i o número de *bits* de R_j;
- $R_{iR_k} := R_{jR_l}$ modifica o R_k-ésimo *bit* menos significativo de R_i, dando-lhe o valor do R_l-ésimo *bit* menos significativo de R_j.

Os programas atómicos permitem alterar directamente a memória do computador, enquanto as primitivas de composição de programas controlam o programa a ser executado a seguir. Basta considerar três maneiras de compor programas:

- $(M_1; M_2)$ (composição sequencial) indica que se deve executar o programa M_1 e de seguida M_2;
- (`if` $(R_i \leq R_j)$ `then` M) (composição alternativa) indica que se deve executar M quando o valor guardado no registo R_i for menor ou igual ao valor guardado em R_j e não executar nada caso contrário;
- (`while` $(R_i \leq R_j)$ `do` M) (composição iterativa) indica que o computador deve executar M enquanto o valor guardado no registo R_i for menor ou igual ao valor guardado em R_j.

O menor conjunto contendo todos os programas atómicos e fechado para a composição sequencial,

[1] Curiosamente, a arquitectura moderna dos computadores baseia-se num modelo que von Neumann propôs para o cérebro humano.

alternativa e iterativa é o conjunto de todos os programas $\mathcal{M}$. Um programa $M \in \mathcal{M}$ induz uma função (parcial) de naturais para naturais. Assuma que a memória inicial do computador é $\xi(x)$. A execução de um programa M a partir desta configuração da memória, se terminar, deixará um valor $f_M(x)$ no registo R_0. Diz-se então que o programa M computa a função parcial $f_M{:}\mathbb{N} \nrightarrow \mathbb{N}$. A função é parcial, pois está indefinida para os naturais que não fazem o programa terminar. Por exemplo, o programa

$$(R_2{:=}3,(\texttt{while}(R_1 \leq R_2)\ do\ R_2{:=}3))$$

induz a função parcial f tal que $f(x)=x$ para $x>3$ e não está definida para $x \leq 3$. Uma função para a qual existe um programa que a induz diz-se *computável*. A generalização para funções com aridade arbitrária é simples, bastando considerar a configuração inicial de memória $\xi(x_0,..., x_{k-1})$ para uma função de aridade k. Após a execução do programa, o resultado deste (caso exista) é o valor que se encontra no registo R_0.

Observe que o conjunto das funções (parciais) de $\mathbb{N}^k$ para $\mathbb{N}$ tem a cardinalidade dos reais, mas o conjunto de todos os programas, isto é, $\mathcal{M}$, tem a cardinalidade dos naturais. Assim, a esmagadora maioria das funções de $\mathbb{N}$ para $\mathbb{N}$ não é computável, e portanto desafia-se o leitor a encontrar uma função que não seja computável.

Um conjunto $A \subseteq \mathbb{N}^k$ diz-se *recursivo* se a sua função característica é computável, isto é, se a função $f{:}\mathbb{N}^k \to \{0,1\}$, tal que $f(\vec{x})=1$ se $x \in A$ é computável.

Para encontrar um conjunto não recursivo usa-se um argumento diagonal à la Cantor. Começa-se por considerar uma enumeração dos programas, isto é, uma bijecção $\gamma{:}\mathbb{N} \to \mathcal{M}$. Dada esta bijecção, é fácil mostrar que o seguinte conjunto

$$H=\{\, x \in \mathbb{N}{:} f_{\gamma(x)} \text{ está definida no valor } x \,\}$$

não é recursivo. Suponha que H é recursivo. Então há um programa M_H que induz a função característica de H e, mais, é possível construir o programa seguinte:

$$M' \equiv M_H;(R_1{:=}1;\ while\ (R_1 \leq R_0)\ do\ R_2{:=}1).$$

Observe que M' não termina se o programa M_H tiver colocado 1 no registo R_0, e termina caso tenha colocado 0 nesse mesmo registo. Com um pouco de esforço verifica-se que $\gamma^{-1}(M') \in H$ se $\gamma^{-1}(M') \notin H$, logo o programa M' não pode existir, e consequentemente a função característica de H não é computável, ou seja, H não é recursivo.

No que se segue, consideram-se apenas funções totais e computáveis. Como seria de esperar, algumas funções são induzidas por programas que utilizam mais tempo e espaço (número de registos e *bits* destes) do que outras. A área da complexidade computacional estuda precisamente os recursos necessários (em tempo e espaço) para programar uma função total e computável. Este texto vai cingir-se ao tempo, pois é o essencial para entender o problema P≠NP. Dada uma configuração de memória do computador ξ, é possível associar a cada programa M o seu tempo de execução $T(\xi,M)$ da seguinte forma:

- $T(\xi,R_i{:=}n) = |n|$;
- $T(\xi,R_i{:=}R_j+R_k) = |R_j^\xi| + |R_k^\xi|$;
- $T(\xi,R_i{:=}|R_j|) = \|R_j^\xi\|$ (o número de *bits* necessário para representar o número de *bits* de R_j^ξ);
- $T(\xi,R_{iR_k}{:=}R_{jR_l}) = R_k^\xi + R_l^\xi$;
- $T(\xi,M_1;M_2) = T(\xi,M_1) + T(\xi',M_2)$ onde ξ' é o estado da memória que se obtém após executar M_1 sobre ξ;
- $T(\xi,\texttt{if}\,(R_1 \leq R_j)\texttt{then}\,M) =$ $\min(|R_i^\xi|, |R_j^\xi|) + \chi(R_i^\xi \leq R_j^\xi) \times T(\xi,M)$ onde $\chi\,(R_i^\xi \leq R_j^\xi)$ toma o valor 1 se $R_i^\xi \leq R_j^\xi$ e 0 caso contrário;
- $T(\texttt{while}(R_i \leq R_j)\,\texttt{do}\,M) = \min(|R_i^\xi|, |R_j^\xi|) + {}$ $+\sum_{v=0}^{k}(T(\xi_v,M) + \min(|R_i^v|, |R_j^v|)))$ onde k é o número de vezes que o ciclo `while` é executado e ξ_v é o estado da memória após a v-ésima execução de M sobre ξ_{v-1} com $\xi_0=\xi$.

Um programa M com entradas diz-se de *tempo polinomial* se existe um polinómio p e natural k tal que, para todo o $n \geq k$,
$\max_{\{x_1,\dots x_k:\sum_{i=0}^{k}|x_i| \leq n\}} T(\xi(x_1,\dots x_k),M) \leq p(n)$.

Se existe um programa de tempo polinomial que induz a função característica de um conjunto $A \subseteq \mathbb{N}^k$, diz-se que o problema da pertença a A é de tempo polinomial. A classe P contém todos os conjuntos cujo problema da pertença é de tempo polinomial. Por exemplo, o subconjunto dos pares está em P (tente encontrar um programa que prova tal facto). Em 2004 [1] foi demonstrado que o conjuntos dos números primos está em P.

Para definir a classe NP vai ser necessário relaxar a definição de função computável em tempo polinomial. Um programa M com entradas $x_1,\dots x_k$, diz-se de *tempo polinomial para os primeiros* $t<k$

argumentos se existe um polinómio p e um natural k tal que para todo o $n \geq k$

$\max_{\{x_1,\ldots x_k:\sum_{i=1}^{t}|x_i|\leq n\}} T(\xi(x_1,\ldots x_k),M) \leq p(n)$.

Um conjunto $A \subseteq \mathbb{N}^k$ diz-se de tempo polinomial não determinístico se existe um programa M com entradas $x_1,\ldots x_k, x_{k+1}$, polinomial, para os primeiros k argumentos, tal que:

- se $(x_1,\ldots x_k) \in A$ então existe $w \in \mathbb{N}$ tal que $f_M(x_1,\ldots x_k, w) = 1$;
- se $(x_1,\ldots x_k) \notin A$, para qualquer $w \in \mathbb{N}$ tem-se $f_M(x_1,\ldots x_k, w) = 0$.

NP é a classe de todos os conjuntos de tempo polinomial não-determinístico[2].

Vale a pena explicar o conceito anterior com um pouco mais de detalhe. Um conjunto de tempo polinomial não determinístico A é um conjunto para o qual, dado um elemento $x \in A$, e na posse de uma certa testemunha w, se consegue verificar em tempo polinomial que $x \in A$. Um exemplo pedagógico de um conjunto em NP é o seguinte:

$F=\{(x,y) \in \mathbb{N}^2: x$ tem factores primos menores que $y\}$.

Se $(x,y) \in F$, e na posse de um factor primo q de x menor que y, podemos determinar em tempo polinomial que $(x,y) \in F$. Para isso basta construir o seguinte programa: (i) testa-se se q é primo; (ii) de seguida, testa-se se q divide x; (iii) e finalmente testa-se se $q<y$. Se todas estas condições se verificarem, o programa retorna 1, caso contrário retorna 0. Este programa é de tempo polinomial, e, se $(x,y) \in F$, existe q tal que para a entrada (x,y,q) o programa retorna 1. Mais, caso $(x,y) \notin F$ não existe testemunha q que faça o programa retornar 1 para a entrada (x,y,q).

Pela definição, obtém-se facilmente que P$\subseteq$NP. Ninguém sabe se o conjunto F se encontra na classe P ou não (e a convicção favorável a um ou outro caso divide a comunidade). Se F está em P, é possível encontrar em tempo polinomial a factorização de um natural em potências de primos, o que teria graves repercussões em criptografia.

Há conjuntos em NP que (quase toda) a comunidade pensa não estarem em P. O problema da pertença de $A \subseteq \mathbb{N}^k$ *reduz-se em tempo polinomial* ao problema da pertença de $B \subseteq \mathbb{N}^l$ se existir uma função computável em tempo polinomial r: $\mathbb{N}^k \rightarrow \mathbb{N}^l$ tal que $x \in A$ sse $r(x) \in B$. Existem conjuntos em NP para os quais todos os restantes problemas NP se reduzem! Estes conjuntos denominam-se *NP-completos*, e basta provar que um problema NP-completo se encontra em P para que P=NP. Existe uma panóplia variada de problemas NP-completos, uns mais famosos que outros. Cook [2] provou que o conjunto das fórmulas proposicionais (devidamente codificadas nos naturais) satisfazíveis, isto é, para as quais existe uma valoração que a satisfaz, é NP-completo.

Há argumentos que indicam que a técnica de diagonalização não será suficiente para demonstrar P≠NP. Muitas tentativas de obter o resultado estão a ser feitas reduzindo o problema a outras áreas da matemática, onde o conhecimento esteja mais avançado. Parece que há muito caminho por desbravar, e, apesar do optimismo de uns, é muito provável que a resposta não seja dada em breve. Em todo o caso, é um problema apaixonante, que fará com certeza suar as gerações vindouras. Para finalizar, o leitor mais interessado poderá encontrar uma exposição de grande qualidade sobre o problema P≠NP no portal do Clay Mathematics Institute da autoria do próprio Stephen Cook. M

Referências

[1] **Manindra Agrawal, Neeraj Kayal e Nitin Saxena** (2004). "Primes is in P". *Annals of Mathematics*, 160:781–793.

[2] **Stephen A. Cook** (1971). "The complexity of theorem-proving procedures". in *STOC*, págs. 151–158.

[3] **Alan Turing** (1936). "On computable numbers, with an application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society, Series 2*, 42:230–265.

[2] A razão de se utilizar a expressão "não determinístico" perde-se ao omitir as máquinas de Turing não determinísticas. Pode-se interpretar o não determinismo como um programa M, utópico, que para uma certa entrada x gera todas as possíveis testemunhas de uma forma não determinada, e retorna 1 se para uma destas testemunhas w a execução do programa retorna $f_M(x,w)=1$ e 0 se para todas as testemunhas $f_M(x,w)=0$.