Algoritmos

META

- Introduzir e medir eficiência de algoritmos.

OBJETIVOS

Ao final da aula o aluno deverá ser capaz de:

- Criar pequenos algoritmos;
- Medir a eficiência de algoritmos simples.

PRÉ-REQUISITOS

- Relação de recorrência (aula 1);

6.1 Introdução

Prezado aluno, passamos da metade de nosso curso, e nesta aula estudaremos uma idéia que talvez tenha sido a chave do desenvolvimento do mundo: os algoritmos. Apesar deles terem tido uma explosão de crescimento com o advento dos computadores, veremos que os algoritmos foram introduzidos na Europa pelo trabalho de Al Khwarizmi, um matemático árabe do século IX, que influenciou Fibonacci, matemático italiano do século XV. Neste famoso trabalho, Al Khwarizmi utilizou o sistema posicional decimal, já conhecido na Índia desde o século VI, que só na época de Fibonacci passou a ser adotado em substituição aos algarismos romanos. Foi nesse sistema posicional decimal que surgiram os primeiros algarismos para adição, multiplicação e divisão de números. Aprenderemos ainda que sempre que temos um algoritmo devemos responder a três perguntas: ele é correto? É eficiente? Pode ser melhorado? Para finalizar, estudaremos a notação O que medirá a eficiência de um algoritmo.

6.2 Livros e Algoritmos

Duas idéias mudaram o mundo. Em 1448 Johann Gutenberg descobriu uma forma de imprimir livros colocando peças metálicas móveis juntas. A literatura se expandiu, a Idade das Trevas terminou, o intelecto humano foi liberado, ciência e tecnologia triunfaram, a Revolução Industrial aconteceu. Muitos historiadores dizem que devemos tudo isso à tipografia. Mas outros insistem que a chave do desenvolvimento não foi a tipografia, mas os algoritmos.

Hoje estamos tão acostumados a escrever os números usando

o sistema decimal que esquecemos que na época de Gutenberg ele escreveu 1448 em numerais romanos. Mas como os romanos (ou europeus do século XV) somavam dois números? Quanto é MCDXLVIII + DCCCXII? Provavelmente só conseguiam adicionar e subtrair pequenos números com o auxílio dos dedos; para coisas mais complicadas tinham que consultar algum especialista em ábacos.

O sistema decimal, inventado na Índia cerca de 600 D.C., foi uma revolução por razões quantitativas: usando apenas 10 símbolos, mesmo números muito grandes podiam ser escritos compactamente e a aritmética podia ser feita eficientemente seguindo passos elementares. Todavia, estas idéias levaram muito tempo para se espalhar, por dificultadas como as barreiras tradicionais do idioma, distância e ignorância. O meio de transmissão mais influente foi um livro texto, escrito em árabe no século IX por um homem que viveu em Bagdá. Al Khwarizmi ensinou os métodos básicos para adição, multiplicação e divisão de números - e até mesmo a extração de raízes quadradas e o cálculo de dígitos de π. Esses procedimentos eram precisos, claros, mecânicos, eficientes e corretos - em resumo, eram algoritmos, um termo em homenagem a este sábio depois que o sistema decimal foi finalmente adotado na Europa, muitos séculos depois.

Desde então, esse sistema posicional decimal e seus algoritmos numéricos têm sido de importância fundamental para as civilizações ocidentais. Eles habilitaram a ciência e a tecnologia; aceleraram a indústria e o comércio. E quando, muito depois, o computador foi finalmente criado, ele encorporou explicitamente o sistema posicional em seus bits. Os cientistas de todo mundo ficaram

ocupados em desenvolver algoritmos cada vez mais complexos para todo tipo de aplicação em problemas e inovações que acabaram mudando o mundo.

6.3 Fibonacci

O trabalho de Al Khwarizmi não teria conseguido uma posição tão segura no ocidente se não fossem os esforços de um homem: o matemático italiano do século XV Leonardo Fibonacci, que viu o potencial do sistema posicional e trabalhou nele para o desenvolver e propagar.

Hoje, Fibonacci é mais conhecido devido à sua famosa sequência de números

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

que é a soma de seus dois últimos predecessores imediatos. Mais formalmente, os números de Fibonacci F_n são gerados por uma regra simples

$$F_n = \begin{cases} F_{n-2} + F_{n-1}, & \text{se} \quad n > 1 \\ 1, & \text{se} \quad n = 1 \\ 0, & \text{se} \quad n = 0 \end{cases}$$

Nenhuma outra sequência de números foi tão estudada ou aplicada em mais campos: biologia, demografia, artes, arquitetura, música, para citar apenas alguns.

Mas qual é o valor preciso de F_{100} ou F_{200}? O próprio Fibonacci queria saber tais coisas. Para responder, precisamos de um algoritmo para calcular o n-ésimo número de Fibonacci.

```
função fib1(n)
se n=0:  retorne 0
se n=1:  retorne 1
retorne fib1(n-1)+fib1(n-2)
```

Sempre que temos um algoritmo, existem três perguntas que devem ser feitas:

1. Ele é correto?
2. Ele é eficiente? Isto é, quanto tempo ele leva, como função de n?
3. Podemos fazer melhor?

A resposta à primeira pergunta é sim, desde que esse algoritmo é precisamente a definição de Fibonacci de F_n. Já a segunda requer um raciocínio mais elaborado. Seja $T(n)$ o número de passos computacionais necessários para calcular $fib1(n)$; o que podemos dizer sobre essa função? Para iniciar, se $n \leq 1$, o procedimento para quase que imediatamente, depois de no máximo dois passos. Então $T(n) \leq 2$ se $n \leq 2$. Para valores maiores de n, existem duas invocações recursivas de `fib1`, levando $T(n-1)$ e $T(n-2)$ passos, mais três passos (cálculo de $n-1$ e $n-2$ e a soma final $fib1(n-1) + fib1(n-2)$). Então,

$$T(n) = T(n-1) + T(n-2) + 3, \text{ para } n > 1$$

Comparando essa à relação de recorrência F_n vemos imediatamente que $T(n) \geq F_n$. Essa é uma notícia muito ruim: o tempo de execução do algoritmo cresce tão rapidamente quanto os números de Fibonacci. $T(n)$ é exponencial em n, o que significa que o algoritmo é impraticável para valores grandes de n. Em resumo, nosso

ingênuo algoritmo recursivo é correto mas incrivelmente ineficiente. O que nos leva à terceira pergunta: podemos fazer melhor?

Um esquema mais sensível poderia estocar os resultados intermediários, $F_0, F_1, \ldots, F_{n-1}$, assim que eles se tornassem conhecidos.

```
função fib2(n)
se n=0:  retorne 0
crie um vetor f[0...n]
f[0]=0,f[1]=1
para i=2...n f[i]=f[i-1]+f[i-2]
retorne f[n]
```

Assim como `fib1`, a correção desse algoritmo é evidente porque usa diretamente a definição de F_n. Quanto tempo ele leva? O loop interno consiste de um passo computacional simples que é executado $n-1$ vezes. Então o número de passos computacionais usado por `fib2` é linear em n.

6.4 Algoritmos Numéricos

Veremos como um algoritmo adequado faz toda diferença.

Exemplo 6.1 (Avaliação de Polinômios)**.** Determinar $p(5)$ para o polinômio $p(x) = 2x^3 - 7x^2 + 4x - 15$.

Solução: Faremos a avaliação por dois algoritmos distintos: método direto e método de Horner (divisão sintética).

(a) **(Método direto)** Ao escrever $p(x) = 2x^3 - 7x^2 + 4x - 15$, entende-se $p(x) = 2.x.x.x - 7.x.x + 4.x - 15$ então:

$$\begin{aligned} p(5) &= 2.5.5.5 - 7.5.5 + 4.5 - 15 && \text{(substituindo x por 5)} \\ p(5) &= 250 - 175 + 20 - 15 && \text{(após 6 multiplicações)} \\ p(5) &= 80 && \text{(após 3 adições)} \end{aligned}$$

Total de operações: 9 (6 multiplicações e 3 adições). Em geral, para avaliar um polinômio $p(x)$ de grau n pelo método direto serão necessárias $\frac{n(n+1)}{2}$ multiplicações e n adições.

(b) **(Método de Horner)** Se reescrevermos $p(x) = 2x^3 - 7x^2 + 4x - 15$ como $p(x) = ((2.x - 7).x + 4).x - 15$ então:

$$\begin{aligned} p(5) &= ((2.5 - 7).5 + 4).5 - 15 && \text{(substituindo x por 5)} \\ p(5) &= (3.5 + 4).5 - 15 && \text{(após 1 multiplicação e 1 adição)} \\ p(5) &= 19.5 - 15 && \text{(após 1 multiplicação e 1 adição)} \\ p(5) &= 80 && \text{(após 1 multiplicação e 1 adição)} \end{aligned}$$

Total de operações: 6 (3 multiplicações e 3 adições). Em geral, para avaliar um polinômio $p(x)$ de grau n pelo método de Horner serão necessárias n multiplicações e n adições. Este método é conhecido como divisão sintética, cujo algoritmo está exemplificado abaixo:

5	2	-7	4	-15
		10	15	95
	2	3	19	80

◄

Exemplo 6.2 (Máximo Divisor Comum)**.** Sejam $a, b \in \mathbb{N}$ com $b < a$. Determine o máximo divisor comum de a e b, $d = MDC(a, b)$.

Solução: Podemos fazer isso de duas maneiras diferentes: pelo método direto e pelo algoritmo da divisão de Euclides.

(a) **(Método direto)** Testamos todos os números de 2 até $a/2$ para acharmos os divisores de a; fazemos o mesmo teste para acharmos os divisores de b; então escolhemos o maior divisor comum. Por exemplo, suponha $a = 258, b = 60$, então denotando D_n o conjunto dos divisores de n temos:

$$\begin{aligned} D_{258} &= \{1, 2, 3, 6, 43, 86, 129, 258\} \\ D_{60} &= \{1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60\} \\ D_{258} \cap D_{60} &= \{1, 2, 3, 6\}(\text{divisores comuns}) \\ MDC(258, 60) &= 6(\text{máximo divisor comum}) \end{aligned}$$

(b) **(Algoritmo de Euclides)** Dividimos a por b e obtemos o resto $r_1 < b$; então dividimos b por r_1 e obtemos o resto $r_2 < r_1$; depois dividimos r_1 por r_2 e obtemos $r_3 < r_2$; prosseguimos dividindo r_{n-2} por r_{n-1} para obtermos o resto $r_n < r_{n-1}$. Esse processo tem fim quando obtemos o resto $r_k = 0$, que sempre ocorre, uma vez que $0 \leq \cdots < r_n < r_{n-1} < \cdots < r_1 < b < a$; aí teremos $r_{k-1} = MDC(a, b)$. Vejamos o exemplo $a = 258, b = 60$. Dividindo 258 por 60, obtemos resto 18; dividindo 60 por 18, obtemos resto 6; dividindo 18 por 6, obtemos resto 0. Portanto, $MDC(258, 60) = 6$. É comum utilizarmos a tabela seguinte quando aplicamos o algoritmo de Euclides:

258	60	18	
60	18	6	0

◄

6.5 Notação O

Suponha que M é um algoritmo e que n seja o tamanho do dado de entrada. A complexidade $f(n)$ de M aumenta quando n aumenta. Normalmente queremos examinar a razão de crescimento de $f(n)$. Em geral, para isso, comparamos $f(n)$ com funções padrão tais como $\ln n, n, n \ln n, n^2, n^3, 2^n$.

A maneira pela qual comparamos a função complexidade $f(n)$ com uma das funções padrão utiliza a notação O, definida a seguir.

Definição 6.1. Sejam $f(x), g(x)$ funções arbitrárias definidas em $U \subset \mathbb{R}$. Dizemos que $f(x)$ é da ordem de $g(x)$, e denotamos isso por

$$f(x) = O(g(x))$$

se existem um número real k e uma constante positiva C tais que:

$$\forall x > k, |f(x)| \leq C|g(x)|$$

Exemplo 6.3 (Polinômio)**.** Suponha que $P(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_m n^m$ tem grau m. Prove que $P(n) = O(n^m)$.

Solução: Seja $b_i = |a_i|$, para $i = 0, 1, \ldots, m$. Então, para $n \geq 1$, temos

$$\begin{aligned} P(n) &= a_0 + a_1 n + a_2 n^2 + \cdots + a_m n^m \\ &\leq b_0 + b_1 n + b_2 n^2 + \cdots + b_m n^m \\ &= \left(\frac{b_0}{n^m} + \frac{b_1}{n^{m-1}} + \cdots + b_m \right) n^m \\ &\leq (b_0 + b_1 + \cdots + b_m) n^m \\ &= C n^m \end{aligned}$$

Portanto, $P(n) = O(n^m)$.

6.6 Conclusão

Nesta aula, vimos que um algoritmo é um procedimento preciso, claro, mecânico, eficiente e correto. Aprendemos que de posse de um algoritmo, devemos responder a três perguntas: ele é correto? Ele é eficiente? Pode ser melhorado? Para finalizar, introduzimos a notação O que compara a complexidade entre algoritmos.

RESUMO

Um algoritmo é um procedimento preciso, claro, mecânico, eficiente e correto.

Sempre que temos um algoritmo, devemos fazer três perguntas:

1. É correto?
2. É eficiente?
3. Pode ser melhorado?

Para medir a complexidade de um algoritmo, utilizamos a notação O. Dizemos que $f(x) = O(g(x))$, se existem um número real k e uma constante positiva C tais que: $\forall x > k, |f(x)| \leq C|g(x)|$.

PRÓXIMA AULA

Na próxima aula partiremos a uma segunda etapa deste curso. Nela, definiremos alguns conceitos elementares da teoria dos grafos e apresentaremos alguns problemas famosos desta teoria. Preparem-se para uma prazerosa sequência de tópicos que vêm sendo alvo de estudo de muitos matemáticos.

ATIVIDADES

ATIVIDADE 6.1. Existe uma maneira mais rápida para calcular o n-ésimo número de Fibonacci do que por fib2? Uma idéia envolve matrizes. Iniciamos escrevendo as equações $F_1 = F_1$ e $F_2 = F_0 + F_1$ que em notação matricial fica:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Similarmente,

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Em geral,

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Então, querendo calcular F_n é suficiente calcular a n-ésima potência da matriz 2×2.

1. Mostre que matrizes 2×2 podem ser multiplicadas usando-se 4 adições e 8 multiplicações. Mas se X é uma matriz 2×2, quantas multiplicações são necessárias para calcular X^n?

2. Mostre que a multiplicação matricial de X^n é da ordem de $O(\log n)$.

ATIVIDADE 6.2. Todo número natural n pode ser escrito na base 2 como segue $n = \sum_{i=0}^{k} j_i 2^i$, denotamos então $n = (i_k, i_{k-1}, \ldots, i_1, i_0)$.

1. Faça um algoritmo que transforma um número natural em binário e vice-versa.

2. Faça um algoritmo que adiciona dois números binários.

ATIVIDADE 6.3. Escreva um algoritmo que dados dois números inteiros a, b determina $d = mdc(a, b)$ e dois inteiros x, y tais que $d = x.a + y.b$.

ATIVIDADE 6.4. Escreva um algoritmo que ordena uma lista com n números inteiros.

ATIVIDADE 6.5. Escreva um algoritmo que, dados dois polinômios $p(x) = \sum_{i=0}^{n} a_i x^i, q(x) = \sum_{i=0}^{m} b_i x^i$, determina o produto $p(x).q(x)$.

ATIVIDADE 6.6. Em cada umas das situações abaixo, indique se $f(n) = O(g(n))$, ou $g(n) = O(f(n))$ ou ambos.

1. $f(n) = n - 100, g(n) = n - 200$

2. $f(n) = n \log n, g(n) = 10n \log 10n$

3. $f(n) = 10 \log n, g(n) = \log(n^2)$

REFERÊNCIAS

DASGUPTA, S., et al. Algorithms. MaGraw-Hill: New York, 2006.

LIPSCHUTZ,S. LIPSON, M. Matemática Discreta. Coleção Schaum. Bookman: São Paulo, 2004.