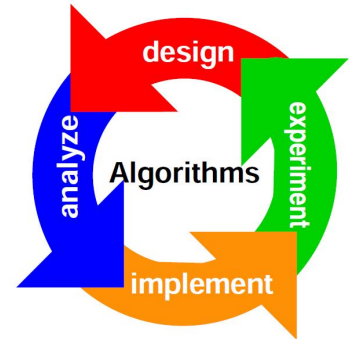

projeto e análise de algoritmos



Esta apostila será utilizada como parte do material didático da disciplina BCC241 (Projeto e Análise de Algoritmos), do curso de Ciência da Computação, oferecida pelo Departamento de Computação da Universidade Federal de Ouro Preto (DECOM-UFOP). Críticas e sugestões podem ser encaminhadas para o autor.

Elton J. Silva
Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro
Ouro Preto - MG, CEP 35400-000
Fone: (31)3559-1640
E-mail: elton@iceb.ufop.br

Capítulo 1

Introdução

Elton J. Silva (eltonsilv@gmail.com)
Departamento de Computação
Universidade Federal de Ouro Preto

“Frequentemente é falado que uma pessoa não entende realmente algo até que ela o ensine a outra pessoa. Na verdade, uma pessoa não entende realmente algo até que ela seja capaz de ensiná-lo a um computador”

Donald E. Knuth, 1974

1. Introdução

A noção de algoritmo é central para toda a Computação. Eles estão presentes nas mais diversas áreas. Aplicações práticas de algoritmos estão em toda parte, por exemplo: criptografia de dados, pesquisa em textos, acesso a banco de dados, otimização combinatória, computação gráfica, etc:

1. O projeto do Genoma Humano que tem o objetivo de identificar 100.000 genes no DNA humano, determinando a sequência de 3 bilhões de pares de bases químicas que constituem o DNA, ordenando essas informações em bancos de dados e desenvolvendo ferramentas para analisá-los. Cada um desses passos requer algoritmos sofisticados.
2. A Internet permite a milhões de pessoas em todo o mundo acessarem e recuperarem grandes quantidades de informação. Para fazer isto, algoritmos eficientes devem ser empregados para gerenciar e manipular grandes volumes de dados. Exemplos de problemas que devem ser resolvidos incluem encontrar boas rotas nas quais os dados devem trafegar, e usar máquinas de busca para encontrar de forma rápida páginas com informações específicas.
3. Comércio eletrônico permite bens e serviços serem negociados eletronicamente. A habilidade para manter informações sigilosas como número de cartões de crédito, senhas, números de contas de banco, são essenciais para a segurança desse tipo de aplicação. Criptografia de chaves públicas e assinaturas digitais estão entre as tecnologias usadas e são baseadas em algoritmos numéricos e teoria dos números.
4. Em diversas situações do cotidiano, é importante alocar recursos escassos da melhor forma possível. Uma companhia de petróleo, por exemplo, que quer saber onde instalar um novo reservatório de forma a maximizar o lucro esperado. Um candidato à presidência pode querer determinar onde gastar mais dinheiro com material publicitário em uma campanha de forma a maximizar as chances de vencer a eleição. Um companhia aérea pode querer alocar a sua frota com o mínimo de custo possível, tendo certeza de que todos os vôos são atendidos e as regulamentações do governo relacionadas ao tráfego aéreo são obedecidas. Todos esses exemplos envolvem algoritmos de otimização combinatória.

O propósito principal do curso de Projeto e Análise de Algoritmos é apresentar como analisar e projetar algoritmos eficientes, através do estudo de uma variedade de algoritmos já bem conhecidos e considerados clássicos na Computação.

O Projeto e a Análise de algoritmos são abordados juntos nesta disciplina: um conhecimento de técnicas de projeto certamente poderá ajudar a criar novos algoritmos, mas sem as ferramentas de análise de algoritmos fica difícil determinar a qualidade dos resultados.

2. Motivação

Imagine a seguinte situação: ordenar um conjunto de um milhão de elementos em uma tabela.

José A. Pressado possui um supercomputador, com velocidade de processamento de 100 milhões de instruções por segundo. Para ordenar o vetor ele contratou um excelente programador para implementar o algoritmo da inserção direta. O algoritmo foi implementado diretamente em linguagem de máquina, e requer $2n^2$ instruções para ordenar n elementos.

João V. Lozz possui um PC XT, com velocidade de processamento de 1 milhão de instruções por segundo. Para ordenar o vetor ele contratou um programador razoável, usando uma linguagem de alto nível com um compilador ineficiente, e o código resultante com $50n \cdot \log_{10} n$ instruções para ordenar n elementos. O algoritmo utilizado foi uma variação do método *quicksort*.

Pergunta-se: Em qual das situações foi gasto o menor tempo para a solução do problema? É fácil verificar que João V. Lozz vai gastar 5 minutos¹, enquanto José A. Pressado gastaria mais de 5 horas. Algoritmos diferentes para resolver o mesmo problema podem diferir dramaticamente em eficiência. Essa diferença pode ser mais significativa que a diferença entre um supercomputador e um PC XT.

Embora muitos alunos não se sintam motivados pela área de projeto e análise de algoritmos, vale a pena persistir. Os benefícios de caráter prático podem, por si só, serem suficientemente compensadores.

A eficiência de um algoritmo geralmente está relacionada à complexidade de tempo e complexidade de espaço de memória utilizado.

a) **Complexidade de tempo:** quanto “tempo” é necessário para computar o resultado para uma instância do problema de tamanho n .

b) **Complexidade de espaço:** quanto “espaço de memória/disco” é preciso para armazenar uma estrutura de dados.

Complexidade de espaço e tempo estão frequentemente relacionadas.

A eficiência computacional medida objetivamente (empiricamente) depende de:

- Como o programador implementou o algoritmo/Estrutura de Dados.
- Características do computador usado para fazer experimentos: velocidade da CPU, capacidade e velocidade de acesso à memória primária/secundária, rede etc.
- Linguagem / Compilador / Sistema Operacional / etc

A medição formal de complexidade de algoritmos tem que ser subjetiva, porém **matematicamente** consistente. Este é o campo da **complexidade assintótica de algoritmos**, que estudaremos no próximo capítulo. Toda essa análise de complexidade tem por objetivo o algoritmo como entidade teórica e não a sua implementação concreta, numa dada linguagem num dado Computador. Assim, são desprezados os tempos envolvidos na execução de operações tais como: chamadas de sub-programas, leitura e escritas, acessos a elementos de vetores ou matrizes, etc.

¹ $(50 \cdot 10^6 \log 10^6)$ instruções / 10^6 instruções/segundos

3. Conceito de algoritmo



De fato, algoritmos já existiam há muitos anos antes da construção do primeiro computador, na década de 40. Uma das versões da origem da palavra algoritmo é que ela foi derivada do nome de um famoso matemático persa Abu Ja'far Mohammed ibn Musa al Khwarizm², que em 835 A.D. escreveu um livro sobre solução de problemas aritméticos.

No Webster, algoritmo é definido como um método para resolver um determinado tipo de problema. Trazendo o conceito mais para a Computação:

- É um método preciso e usável por um computador para a solução de um problema.
- É composto por um conjunto finito de passos, cada um requerendo uma ou mais instruções.
- Cada operação deve ser perfeitamente definida e clara.

É importante lembrar que para um mesmo problema podem existir vários algoritmos distintos que o resolva. Esses, por sua vez, se distinguem quanto à complexidade em que são formulados. Essa complexidade geralmente está associada a tempo de execução e espaço de memória utilizado. Cabe-nos, sempre que possível, escolher aqueles que gastam menos tempo e consomem menos espaço de memória.

É importante lembrar que para um mesmo problema podem existir vários algoritmos distintos que o resolva. Esses, por sua vez, se distinguem quanto à complexidade em que são formulados. Essa complexidade geralmente está associada a tempo de execução e espaço de memória utilizado. Cabe-nos, sempre que possível, escolher aqueles que gastam menos tempo e consomem menos espaço de memória.

É bom relembrar que algoritmo difere de programa. Enquanto os algoritmos são formulações em níveis mais altos de abstração, independentes de uma linguagem de programação específica, os programas de computador são formulações concretas de algoritmos abstratos baseados em uma representação particular dos dados, e escritos em uma linguagem de programação específica, por exemplo: Java, C++, Haskell, Prolog, PHP etc.

4. Áreas de abrangência

O estudo de algoritmos pode ser subdividido em cinco áreas importantes e distintas de pesquisa:

1. Como **analisar algoritmos**: a análise de algoritmos refere-se ao processo de determinar o tempo e a memória de um computador que o algoritmo vai requerer. Um estudo como este nos permite fazer um julgamento qualitativo de um algoritmo sobre outro e prever se o software terá alguma restrição de eficiência, considerando o pior caso, o caso médio e o melhor caso.

² Literalmente significa "da cidade de Khwarizmi. Esta cidade é hoje conhecida como Khiva e está localizada na ex União Soviética.

2. Como **verificar a corretude de algoritmos**: é o processo no qual se verifica se o algoritmo fornece saída correta para toda entrada de dados possível. Esta verificação deve ser feita independente do código de programação utilizado na implementação.

3. Como **projetar algoritmos**.

4. Como **expressar algoritmos**: a programação estruturada, por exemplo, tem como finalidade a expressão clara e concisa de um algoritmo em linguagem de programação.

5. Como **testar programas**: consiste em duas fases, depuração e desempenho. A **depuração** consiste em executar o programa sobre um conjunto de dados para verificar se ele funciona corretamente ou se algum resultado falho ocorre. "Uma prova de corretude é mais válida que mil testes". O **desempenho** é o processo de executar o programa correto sobre um conjunto de dados para medir o tempo e o espaço gastos para obter o resultado desejado.

5. Processo de desenvolvimento de programas

Desenvolver um programa é mais que simplesmente sentar-se à frente do computador e começar a escrever algumas linhas de código numa determinada linguagem de programação, como por exemplo, Java ou C++.

O processo de desenvolvimento completo de programas é composto basicamente pelos seguintes passos:

1. Estabelecimento do problema
2. Desenvolvimento de um modelo
3. Projeto do algoritmo
4. Corretude do algoritmo
5. Análise de complexidade do algoritmo
6. Implementação (tradução do algoritmo para uma linguagem de programação específica)
7. Testes do programa: depuração e desempenho
8. Documentação

Vamos exemplificar esses passos com um problema bastante conhecido: encontrar o máximo divisor comum³ entre dois números inteiros a e b .

1. **Estabelecimento do problema**: Antes de mais nada, deve-se entender o enunciado do problema que se está tentando resolver. Encontrar o MDC de dois números a e b significa encontrar o maior número que divide simultaneamente a e b . Por exemplo, o maior divisor comum entre os números 32 e 24 é 8, entre os números 25 e 12 é 1.

³ O cálculo de MDC entre números inteiros é uma parte importante de algoritmos de criptografia de dados. Por exemplo, o famoso algoritmo de criptografia RSA, criado por Ron Rivest, Adi Shamir e Len Adlema (RSA) em 1978, até hoje é muito utilizado, principalmente em conexões seguras na Internet.

2. **Desenvolvimento de um modelo:** para o caso do MDC alguns modelos conhecidos são (i) o modelo de decomposição em fatores primos e (ii) o modelo de Euclides.

(i) No modelo de decomposição em fatores primos, se ainda me lembro dos meus tempos de cursinho pré-vestibular, os números a e b são decompostos em fatores primos, por exemplo: $32 = 2^5$, e $24 = 2^3 \times 3$. O MDC (a, b) será o produto dos fatores primos comuns com menor expoente, no caso, 2^3 . Outro exemplo, o MDC entre 120 e 72: $120 = 2^3 \times 3 \times 5$, $72 = 2^3 \times 3^2$. O MDC $(120, 72) = 2^3 \times 3 = 8 \times 3 = 24$.

(ii) No modelo de Euclides, ou método das divisões sucessivas, dividimos o maior número pelo menor. Se a divisão for exata, o MDC entre os números a e b será o menor deles. Se a divisão não for exata, dividimos o menor número pelo primeiro resto. Se essa nova divisão também não for exata, dividimos o primeiro resto pelo segundo. Efetuamos, assim, sucessivas divisões até obter zero. O m.d.c é o último divisor. Por exemplo, o $\text{MDC}(150, 65) = 5$:

Quociente	2	3	4	
150	65	20	5	Último Divisor
20	5	0		Resto
Restos				

3. **Projeto do algoritmo:** depende do modelo escolhido no passo 2. Por exemplo, para o modelo de Euclides podemos ter o seguinte projeto de um algoritmo iterativo em alto nível:

MDC(a, b):

```
{ r ← (a mod b); // o resto da divisão inteira de a por b
  enquanto (r ≠ 0) faça
    { a ← b; b ← r; r ← (a mod b) }
  retorne b;
}
```

Observe que poderíamos também ter optado pelo projeto de um algoritmo recursivo⁴.

4. Corretude do algoritmo

Esta é uma das etapas mais difíceis. Provar a corretude de um algoritmo é fazer uma prova formal de que o algoritmo funciona para qualquer entrada. Foge do escopo deste curso apresentar uma prova de corretude para o algoritmo **MDC**.

5. Análise de complexidade do algoritmo

⁴ Recursão é um recurso poderoso na Computação e refere-se genericamente a um procedimento ou função que chama a si mesmo direta ou indiretamente.

A análise de complexidade deve levar em consideração o melhor caso, pior caso e caso médio. Por exemplo, se os valores iniciais são iguais, $a = b$, o algoritmo termina rapidamente. Dá para perceber que a complexidade está associada aos valores de entrada a e b .

- 6. Implementação** (tradução do algoritmo para uma linguagem de programação específica, por exemplo: C, C++, Java, Pascal, Haskell etc). A seguir, um trecho do código escrito na linguagem C++:

```
int MDC(int a, int b)
{
    int r;
    r = a%b;
    while (r != 0)
        { a = b; b = r; r = a%b; }
    return b;
}
```

7. Testes do programa: depuração e desempenho

Depurar o programa de MDC consiste em testá-lo com diferentes valores de entrada para a e b (números negativos, positivos, grandes, pequenos, etc.) e observar se os resultados obtidos são corretos.

Testar o desempenho do programa MDC deve levar em consideração a máquina onde o programa vai ser executado. Devem ser feitas tomadas de tempo para alguns casos, consumo de memória para as chamadas recursivas.

8. Documentação

A documentação deve fazer parte de todo o processo de desenvolvimento de programas. No caso do problema do mdc, todos os passos anteriores devem ter sido devidamente documentados para facilitar futuras modificações pelo autor (ou por outras pessoas que queiram melhorar o programa).

6. Visão Geral do curso BCC241

Algumas perguntas que surgem quando nos deparamos com um problema a ser resolvido no computador são:

- *Como represento a informação/dados?*
- *Como manipulo/acesso esses dados de forma eficiente?*
- *Existe um algoritmo mais eficiente que outro para resolver este problema?*
- *Existe uma estrutura dos dados (forma como são representados) que é mais eficiente que outra?*

O conteúdo que veremos em BCC241 nos ajudará a responder a essas perguntas futuramente. O curso começa com uma parte mais voltada para ANÁLISE DE ALGORITMOS, ou seja, como escolher um algoritmo para resolver um problema? Qual algoritmo gasta menos tempo? Qual algoritmo consome menos espaço de memória? Para isto, apresentaremos noções de COMPLEXIDADE DE ALGORITMOS e algumas FERRAMENTAS MATEMÁTICAS que nos auxiliarão a proceder esta análise.

Numa segunda parte do curso, apresentaremos diferentes TÉCNICAS DE PROJETO DE ALGORITMOS, com particular atenção para os métodos de DIVISÃO E CONQUISTA, GULOSO, PROGRAMAÇÃO DINÂMICA, BACKTRACKING E BRANCH&BOUND. Para cada uma dessas técnicas veremos exemplos de problemas que podem ser resolvidos com a sua utilização e as complexidades dos algoritmos correspondentes.

Bibliografia

Algoritmos: Teoria e Prática, T. H. Cormen, C. E. Leiserson, R. L. Rivest & C. Stein, Editora Campus, 1991.

Computer Algorithms, E. Horowitz, S. Sahni & S. Rajasekaran, Computer Science Press, 1998

Exercícios de Fixação

*Se ouço, esqueço.
Se vejo, relembro.
Se faço, entendo.
-- Provérbio Chinês.*

1) Problema de Multiplicação de Inteiros à la russe

Sejam n e m dois inteiros positivos cuja multiplicação desejamos calcular. A tabela a seguir ilustra a “execução” do método de la russe aplicado aos inteiros 50 e 65.

50	65	-
25	130	130
12	260	-
6	520	-
3	1040	1040
1	2080	2080
		3450

Discuta os passos do processo de desenvolvimento de programas (seção 6) aplicados ao Problema de Multiplicação à la russe.

- 2) Explique a diferença entre prova de corretude, teste de depuração e teste de desempenho. Exemplifique.
- 3) Elabore um algoritmo que, dado um conjunto não ordenado C com n números reais e um real x , determine se existem dois números a e b pertencentes a C , tais que $a + b = x$. Compare a eficiência do seu algoritmo com o de seus colegas.