Universidade Federal de Ouro Preto
PCC104 - Projeto e Análise de Algoritmos
Semanas 1 e 2

Prof. Rodrigo Silva

August 9, 2021

Natália Fernanda de Castro Meira

1 Leitura Recomendada

- Capítulo 1 - Introduction to the Design and Analysis of Algorithms (3rd Edition) - Anany Levitin
- Capítulo 2 - Introduction to the Design and Analysis of Algorithms (3rd Edition) - Anany Levitin

2 Questões (discussões na aula 11/08/2021)

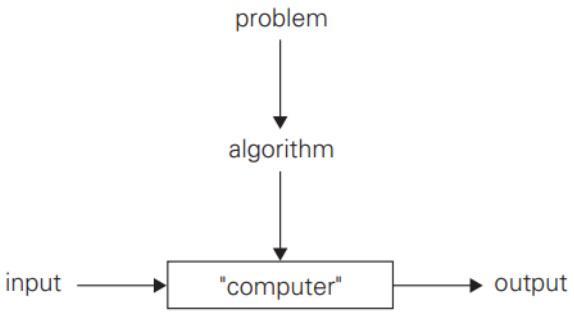
1. Por quê estudamos algoritmos?

Estudamos os algoritmos pois os algoritmos são muito mais do que uma ferramente para se chegar a uma resposta. Os algoritmos compreendem o cerne da ciência da computação, pois desenvolvem a capacidade e habilidade análitica de quem o estuda, uma vez que, pode-se dizer que ninguém realmente aprende antes de ensinar a alguém e, no caso dos algoritmos, ninguém realmente aprende antes de ensinar a um computador.

Ao se estudar os algoritmos, desenvolve-se tecnicas e estratégias para abordar determinado problema e chegar a uma solução. É preciso pensá-lo, escrevê-lo, compreendê-lo e analisá-lo. Assim, a solução de problemas pode ser compreendida de maneira muito mais profunda do que quando estudamos de maneira tradicional.

2. O que é um algoritmo?

Para o autor Anany Levitin, "um algoritmo é uma sequência de instruções claras para resolver um problema, ou seja, para obter uma saída necessária para qualquer entrada legítima em um período de tempo finito"



No caso de instruções, espera-se que sejam de tal forma que, algo ou alguém seja capaz de compreendê-las e segui-las. A implementação de um algoritmo não depende da suposição de que sera destinado a um computador. Antes dos

computadores que conhecemos, a palavra computador estava relacionada a uma pessoa que executava cálculos numéricos. Alguns pontos importantes são:

- Atender ao requisito de não-ambiguidade em todas as etapas;
- O intervalo de entradas para o qual o algoritmo funciona deve ser especificado;
- O mesmo algoritmo pode ser representado de várias maneiras diferentes;
- Podem existir diversos algoritmos para se resolver o mesmo problema.
- Algoritmos para o mesmo problema podem ser baseados em ideias muito diferentes e podem resolver o problema com velocidades drasticamente diferentes.

3. Considere o algoritmo de Euclides (https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations) para o cálculo do maior divisor comum. Como sabemos que este termina em tempo finito?

Para cada atualização do algoritmo, ou seja, considerando uma iteração gcd(m,n)=gcd(n, m mod n), $m_{k+1} < m_k$ e $n_{k+1} < n_k$, ou seja, o segundo inteiro diminui a cada iteração e não pode ser negativo. Assim, eventualmente, ele se torna zero e o algoritmo para. Então, é um algoritmo finito.

4. Considere o seguinte procedimento para o cáculo do maior divisor comum:

STEP 1 Find the prime factors of m.

STEP 2 Find the prime factors of n.

STEP 3 Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If p is a common factor occurring p_m and p_n times in m and n, respectively, it should be repeated $min\{p_m, p_n\}$ times.)

STEP 4 Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

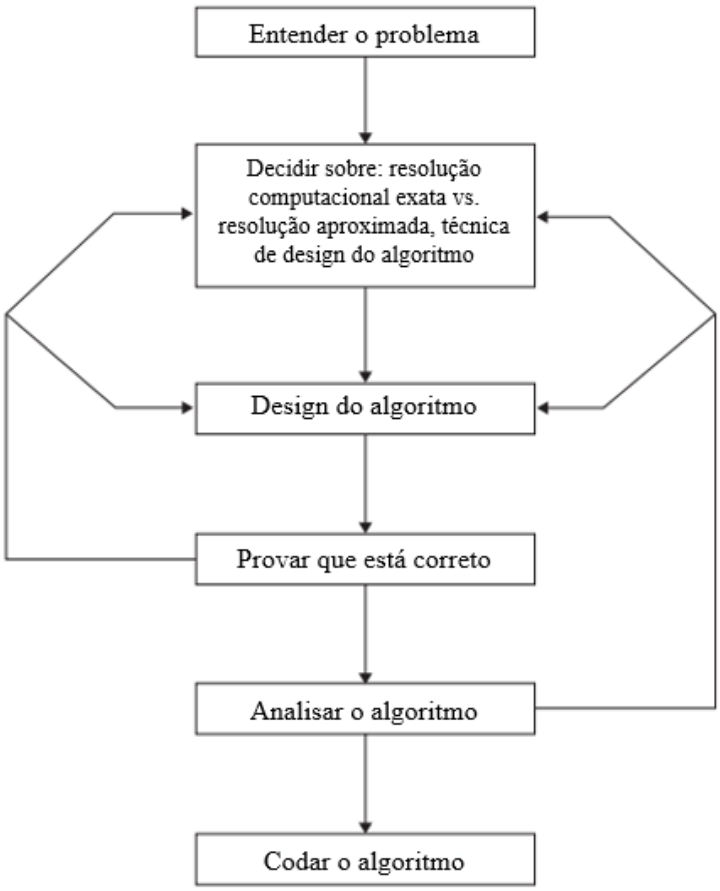
Por quê este procedimento não se qualifica como um algoritmo legítimo?

Os motivos pelos quais este procedimento não se qualifica como um algoritmo são:

- As entradas não foram especificadas;
- As instruções para o cálculo da fatoração de número primos não foram especificadas nem como obter a lista de número primos;
- As instruções para o cálculo do maior dividor comum dos número não foram especificadas;
- O procedimento possui eficiencia inferior ao método de Euclides.

5. Ilustre, com um fluxograma, o processo de projeto de análise de algoritmos e apresente uma pequena explicação sobre cada passo.

O fluxograma a seguir apresenta uma sequência de etapas que normalmente percorremos ao projetar e analisar um algoritmo:



1. Entender o problema: compreender completamento o problema dado, descrevendo-o cuidadosamente e fazendo perguntas para que todas as dúvidas sejam sanadas;
2. Nem todos os problemas serão resolvidos com uma resposta exata, pois podem envolver raízes quadradas, equações lineares, etc., ou pela própria complexidade do problema. Qual o dispositivo que o algoritmo será executado, disponibilidade da capacidade computacional. Qual a técnica.
3. Como organizar e classificar a solução deste algoritmo. Especificar os passos/etapas. Ser capaz de escrever o peseudocódigo do seu algoritmo. Pseudocódigo é uma mistura de uma linguagem natural e construções semelhantes à linguagem de programação. Estrutura de dados. Sequencia de passos. Linguagem de programação.
4. Depois, provar a corretude do algoritmo, que ele produz um resultado para cada entrada em uma quantidade finita de tempo (teste).
5. Analisar a eficiência de tempo (velocidade de execução) e eficiência de espaço (quanta memória ele ocupa).
6. Implementar o algoritmo em um programa de computador.

Como regra, um bom algoritmo é o resultado de esforços repetidos e retrabalho.

2 Questões (discussões na aula 16/08/2021) (até ex 23)

6. Indique como a estrutura de dados abstrata, Fila de Prioridade pode ser implementada com as seguintes estruturas de dados.

Fila de prioridade: estrutura de dados que exigem a seleção de um item de prioridade mais alta entre um conjunto de candidatos que muda dinamicamente.

Estrutura de dados: um esquema para armaazenar dados.

Estrutura de dados Lineares

Arranjo ou array: uma sequencia de n itens do mesmo tipo, que são armazenados de forma contigua na memória do computador, estrutura unidimensional. Para identificar o item precisamos de um número.

Lista encadeada (linked list): Sequência de nós. Nó é um par que contém um dado e uma referência para o próximo nó. Não tem acesso aleatório. Para acessar o segundo elemento, tem que percorrer todos (no caso o primeiro).

Estruturas lineares abstratas

Pilhas (stack) : só pode inserir e retirar dados do topo (LIFO)

Fila (qeuer) : inserir no final da fila e retirar do inicio da fila (FIFO)

Filha de prioridades (Priority Quer) : cada elemento tem um prioridade e só pode retirar o elemento de maior prioridade.

Arvore de busca binária: grafo acíclico conectado. Grafo definido pelo par de dois conjuntos: G=(V, E), vérticas e arestas. Um grafo conectado é um grafo em que para todo par de vértices u,v existe um caminho de u para v. Arvore binária: cada vértice tem apenas dois filhos.

(a) Um arranjo (array) não ordenado.
(b) Um arranjo (array) ordenado.
(c) Árvore de busca binária.

7. Como você implementaria um dicionário de tamanho pequeno, n = 50, sabendo que todos os elementos são distintos. Especifique a implementação de cada operação do tipo de dados abstrato Dicionário.

PASSO 1 – Criar um dicionários contendo n=50 elementos (array)

PASSO 2 – Criar um loop, de 0 até (n-1), onde, para cada elemento, atribui-se um objeto (array).

PASSO 3 – Criar uma entrada para o usuário: "Digite o número do elemento:" (string c entrada de um inteiro)

PASSO 4 – Retorna o objeto do dicionario. (searching problem)

8. Para cada uma das seguintes operações, indique qual a estrutura de dados mais adequada.

(a) Atender chamadas telefônicas em ordem de prioridade.

Array ordenado

(b) Enviar uma sequência de pedidos a consumidores na ordem em que eles são recebidos.
Fila – implementada por arranjo
(d) Implementar uma calculadora para realizar operações aritméticas simples.
Arvores

9. A o quê se referem os termos Complexidade de Tempo e Complexidade de Espaço?
A eficiência de tempo, também chamada de complexidade de tempo, indica a rapidez com que um algoritmo em questão é executado. A eficiência de espaço, também chamada de complexidade de espaço, refere-se à quantidade de unidades de memória exigidas pelo algoritmo, além do espaço necessário para sua entrada e saída (p.42).

10. Defina os termos, eficiência de melhor caso, caso médio e pior caso.

A eficiência de pior caso de um algoritmo é sua eficiência para a entrada de pior caso de tamanho n, que é uma entrada (ou entradas) de tamanho n para a qual o algoritmo é executado por mais tempo entre todas as entradas possíveis desse tamanho. A maneira de determinar a eficiência do pior caso de um algoritmo é, em princípio, bastante simples: analise o algoritmo para ver que tipo de entradas produzem o maior valor da contagem da operação básica C (n) entre todas as entradas possíveis de tamanho n e em seguida, calcule esse valor de pior caso Cworst (n).
A eficiência do melhor caso de um algoritmo é sua eficiência para a entrada do melhor caso de tamanho n, que é uma entrada (ou entradas) de tamanho n para a qual o algoritmo executa o mais rápido entre todas as entradas possíveis desse tamanho. Conseqüentemente, podemos analisar a eficiência do melhor caso da seguinte maneira. Primeiro, determinamos o tipo de entradas para as quais a contagem C (n) será a menor entre todas as entradas possíveis de tamanho n. (Observe que o melhor caso não significa a menor entrada; significa a entrada de tamanho n para a qual o algoritmo é executado mais rápido.) Em seguida, verificamos o valor de C (n) nessas entradas mais convenientes. Por exemplo, as entradas de melhor caso para pesquisa sequencial são listas de tamanho n com seu primeiro elemento igual a uma chave de pesquisa; consequentemente, Cbest (n) = 1 para este algoritmo.
A eficiência do caso médio busca fornecer as informações necessárias sobre o comportamento de um algoritmo em uma entrada "típica" ou "aleatória". Para analisar a eficiência de caso média do algoritmo, devemos fazer algumas suposições sobre possíveis entradas de tamanho n.

11. Qual, ou quais os problemas de utilizar unidades de tempo, por exemplo, segundos para analisar o tempo de execução de algoritmos? Qual a a estratégia mais adequada para esta tarefa?

Os problemas de utilizar unidades de tempo, por exemplo, segundos para analisar o tempo de execução de algoritmos são:

- dependência da velocidade de um determinado computador;
- dependência da qualidade de um programa que implementa o algoritmo, e;
- do compilador usado na geração do código de máquina e a dificuldade de cronometrar o tempo de execução real do programa.

A estratégia mais adequada para esta tarefa é identificar a operação mais importante do algoritmo, chamada de operação básica, a operação que mais contribui para o tempo total de execução, e calcular o número de vezes que a operação básica é executada. Assim, a estrutura estabelecida para a análise da eficiência de tempo de um algoritmo sugere medi-la contando o número de vezes que a operação básica do algoritmo é executada em entradas de tamanho n.

12. Para cada uma das seguintes funções, indique quanto o valor da função aumenta se o tamanho do argumento aumentar 4 vezes.

a) $\log_2 n = 2$
b) $\sqrt{n} = 2$
c) $n = 4$
d) $n^2 = 16$
e) $n^3 = 64$
f) $2^n = 2^{3n}$

13. Eliminação Gaussina é um algoritmo clássico para resolver um sistema de n equações lineares com n variáveis. O método requer aproximadamente 1/3 n^3 multiplicações que é a operação básica do algoritmo.
(a) Quantas vezes mais devagar você espera que a resolução de um sistema de 1000 equações seja em relação a um sistema de 500.
8x

(b) Você está considerando comprar 1000 vezes mais rápido do que o seu atual. Por qual fator o novo computador irá aumentar o tamanho dos sistemas resolvíveis no antigo dada a mesma quantidade de tempo?
10x

14. Descreva as notações $\mathbf{O}, \mathbf{\Omega}$ **e** $\mathbf{\Theta}$.

$\mathbf{O}$ - Informalmente, O (g (n)) é o conjunto de todas as funções com uma ordem de crescimento inferior ou igual a g (n) (para dentro de um múltiplo constante, pois n vai para o infinito).
Uma função t (n) é dita em O (g (n)), denotada t (n) ∈ O (g (n)), se t (n) é limitado acima por algum múltiplo constante de g (n) para todo n grande, isto é, se existe alguma constante c positiva e algum número inteiro não negativo n0 tal que

$$t(n) \leq cg(n) \text{ for all } n \geq n0$$

$\mathbf{\Omega}$ - representa o conjunto de todas as funções com uma ordem de crescimento superior ou igual a g (n) (para dentro de um múltiplo constante, conforme n vai para o infinito).
Uma função t (n) é dita em $\mathbf{\Omega}$ (g (n)), denotada t (n) ∈ $\mathbf{\Omega}$ (g (n)), se t (n) é limitado abaixo por algum múltiplo constante positivo de g (n) para todo n grande, isto é, se existir alguma constante c positiva e algum número inteiro não negativo n0 tal que

$$t(n) \geq cg(n) \text{ for all } n \geq n0$$

$\mathbf{\Theta}$ - o conjunto de todas as funções que têm a mesma ordem de crescimento de g (n) (para dentro de um múltiplo constante, conforme n vai para o infinito).
Uma função t (n) é dita em (g (n)), denotada t (n) ∈ $\mathbf{\Theta}$ (g (n)), se t (n) é limitado acima e abaixo por alguns múltiplos constantes positivos de g (n) para todos os n grandes, ou seja, se existem algumas constantes positivas c1 e c2 e algum número inteiro não negativo n0 tal que

$$c2g(n) \leq t(n) \leq c1g(n) \text{ for all } n \geq n0.$$

15. Prove o seguinte teorema:
TEOREMA: Se t1(n) ∈ O(g1(n)) e t2(n) ∈ O(g2(n)) então t1(n) + t2(n) ∈ O(max{g1(n); g2(n)})

(OBS: Afirmações análogas são verdadeiras para $\boldsymbol{\Omega}$ e $\boldsymbol{\Theta}$).

No caderno.

16. Utilize limites para comparar as seguintes ordens de crescimento:

No caderno.

a) ½ n(n-1) e n^2

b) $\boldsymbol{\log_2 n}$ e $\boldsymbol{\sqrt{n}}$

c) N! e $\boldsymbol{2^n}$

17. Utilize as definições informais de $\boldsymbol{O, \Omega}$ e $\boldsymbol{\Theta}$. para determinar quais das afirmações abaixo são verdadeiras e quais são falsas.

No caderno.

a) n(n+1)/2 ∈ $\boldsymbol{O}$ (n^3)

b) n(n+1)/2 ∈ $\boldsymbol{\Theta}$ (n^3)

c) n(n+1)/2 ∈ $\boldsymbol{O}$ (n^2)

d) n(n+1)/2 ∈ $\boldsymbol{\Omega}$ (n)

18. Prove que todo polinômio de grau k, $\boldsymbol{p(n) = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} \ldots + a_0}$ com ai > 0, pertence a $\boldsymbol{\Theta(n^k)}$ (Dica? Você pode provar esta afirmação utilizando limites.)

No caderno.

19. Prove que funções exponenciais, a^n, têm diferentes ordens de crescimento para diferentes valores da base a > 0. (Analise o limite, $\boldsymbol{\lim_{n\to\infty} \frac{a_1^n}{a_2^n}}$)

No caderno.

20. Considere os três algoritmos abaixo:

ALGORITHM *Mystery*(n)

//Input: A nonnegative integer n

$S \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

$S \leftarrow S + i * i$

return S

1

ALGORITHM *Secret*$(A[0..n-1])$

//Input: An array $A[0..n-1]$ of n real numbers

$minval \leftarrow A[0]$; $maxval \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

if $A[i] < minval$

$minval \leftarrow A[i]$

if $A[i] > maxval$

$maxval \leftarrow A[i]$

return $maxval - minval$

2

ALGORITHM *Enigma*$(A[0..n-1, 0..n-1])$

//Input: A matrix $A[0..n-1, 0..n-1]$ of real numbers

for $i \leftarrow 0$ **to** $n-2$ **do**

for $j \leftarrow i+1$ **to** $n-1$ **do**

if $A[i, j] \neq A[j, i]$

return false

return true

3

Para cada algoritmo responda:

(a) O que este algoritmo computa?

No caderno.

(b) Qual a operação básica deste algoritmo?

No caderno.

(c) Quantas vezes esta operação básica é executada?

No caderno.

(d) Qual a classe deste algoritmo em relação a eficiência?

No caderno.

(e) Sugira alguma melhora ou um novo algoritmo melhor e indique a classe desta sugestão. Se você não conseguir, tente provar que, de fato, a melhora não pode ser feita.

No caderno.

24. Quais são os limites da análise matemática de algoritmos? Qual a alternativa?

Mesmo alguns algoritmos aparentemente simples provaram ser muito difíceis de analisar com precisão e certeza matemáticas. Aplicabilidade difícil no caso médio. So analisamos o algoritmo em termos do tamanho da entrada. Mas dependendo dos niveis de desorganização dos dados em termos de uma ordenação, pode ser mais rapido ou não. Ou seja. Não so o tamanho da entrada, mas também as distribuições da entradas. E temos que levar todas essas possibilidades em consideração.

A principal alternativa para a análise matemática da eficiência de um algoritmo é sua análise empírica. Testar diversas entradas e avaliar o comportamento. Levar em consideração nos experimentos quais os tipos de entradas são relevantes, a métrica (tempo, por exemplo), saber a complexidade (expressão que define o custo em função do tamanho da etrada, plotar isso se for dificil matematicamente).

25. Resuma o processo de análise empírica de algoritmos, descrito na seção 2.6 do Capítulo 2 do livro Introduction to the Design and Analysis of Algorithms (3rd Edition) - Anany Levitin.

Plano Geral para a Análise Empírica da Eficiência do Algoritmo no Tempo

1. Compreenda o propósito do experimento – ditar como a eficiência do algoritmo deve ser medida.
2. Decida a métrica de eficiência M a ser medida e a unidade de medida (uma contagem de operação vs. uma unidade de tempo).
3. Decida sobre as características da amostra de entrada (seu intervalo, tamanho e assim por diante).
4. Prepare um programa implementando o algoritmo (ou algoritmos) para a experimentação.
5. Gere uma amostra de entradas.
6. Execute o algoritmo (ou algoritmos) nas entradas da amostra e registre os dados observados.
7. Analise os dados obtidos.