Respostas do capítulo 1 do livro: *Introduction to Algorithms*, de Cormen, Thomas H. et al. (3ª ed., 2009)

Abrantes Araújo Silva Filho

2018-01

Sumário

1 O que é este documento?

Este documento contém as minhas respostas aos exercícios e problemas do capítulo 1 do livro *Introduction to Algorithms*, de Cormen, Thomas H. et al. (3ª ed., de 2009), que utilizei na disciplina de Algoritmos durante minha gradução em Ciência da Computação.

ATENÇÃO: não garanto que tudo aqui está correto, pelo contrário, algumas respostas expressam minha visão particular e podem estar em desacordo com a "resposta padrão" dos autores do livro ou do professor da disciplina de Algoritmos. Também não garanto que todos os exercícios e problemas do capítulo estarão resolvidos aqui.

De qualquer modo, se você quiser utilizar este documento como base para seu próprio estudo, tenha em mente o seguinte:

ESTE DOCUMENTO É FORNECIDO "NO ESTADO EM QUE SE ENCONTRA", SEM GARANTIAS DE QUALQUER NATUREZA, EXPRESSAS OU IMPLÍCITAS. EM NENHUMA HIPÓTESE O AUTOR PODERÁ SER RESPONSABILIZADO POR QUALQUER RECLAMAÇÃO, DANOS OU OUTROS PROBLEMAS DECORRENTES DO USO DESTE CONTEÚDO.

Este documento (em formato PDF) e o original em LaTeX estão disponíveis no seguinte repositório GitHub: https://github.com/abrantesasf/algoritmos

2 Exercícios

2.1 Grupo 1.1:

Exercício 1.1-1 Um exemplo real da necessidade de algoritmos que necessitam de ordenação (sorting) é a ordenação alfabética de uma lista de palavras para a criação de um dicionário.

Ou a criação de um ranking nacional com as notas finais de todos os estudantes brasileiros que participaram do Enem de 2017.

Exercício 1.1-2 Além da velocidade (tempo de execução), outras medidas de eficiência de um algoritmo podem incluir:

- Uso de memória
- Uso de IO de disco
- Uso de banda de rede
- Uso de random bits

Exercício 1.1-3 Uma estrutura de dados que já vi antes é o *vetor*, que é um array unidimensional de n números. É bom para cálculos, mas só armazena dados de um mesmo tipo.

Exercício 1.1-4 O problema da menor distância entre dois pontos em um mapa e o problema do caixeiro viajante são semelhantes no sentido de que ambos os problemas tratam de distâncias a serem percorridas, mas são muito diferentes quanto a complexidade da tarefa computacional. Encontrar somente a menor distância entre dois pontos pode ser resolvido algoritmicamente com eficiência, ao passo que o problema do caixeiro viajante é um problema NP-completo, ou seja, ainda não existe uma solução ótima eficiente para sua resolução.

Exercício 1.1-5 Um problema na qual apenas a melhor solução é aceitável seria, por exemplo, a identificação de um possível objeto voador que vem em nossa direção como um ICBM atômico ou outro objeto qualquer: identificar erroneamente um objeto como ICBM atômico pode iniciar uma guerra nuclear, portanto o algoritmo de identificação deve ser ótimo.

Um problema no qual uma solução próxima da melhor é aceitável seria, por exemplo, determinar a raiz quadrada de um número com 30 casas decimais.

2.2 Grupo 1.2:

Exercício 1.2-1 Um exemplo de aplicação que requer o uso de algorítmos no nível da aplicação é um serviço web que determina como viajar de uma localização à outra: seriam necessários algoritmos para determinar o caminho mais curto (ou outro tipo de caminho especificado pelo usuário, tais como rotas sem pedágio), a renderização de mapas e a interpolação de endereços.

Exercício 1.2-2 Para algoritmos de insertion sort que rodam em $8n^2$ passos e algoritmos de merge sort que rodam em $64n \lg n$ passos, o insertion sort será mais rápido do que o merge sort quando:

$$\begin{aligned} 8n^2 &< 64n \lg n = \\ n^2 &< 8n \lg n = \\ n &< 8 \lg n \end{aligned} \tag{1}$$

Assim, sempre que $n < 8 \lg n$, onde lg é o logarítmo de base 2, o insertion sort será mais rápido do que o merge sort. Fazendo uma rápida tabela, podemos constatar que isso somente ocorre quando $2 \leq n \leq 43$. Para qualquer $n > 43$, o merge sort será mais rápido do que o insertion sort, nos tempos de execução informados pelo problema.

1.2-3 O menor valor de n para que um algoritmo cujo tempo de execução é de $100n^2$ rode mais rápido do que outro algoritmo com tempo de execução 2^n na mesma máquina,

$$100n^2 < 2^n \tag{2}$$

também pode ser obtido com uma planilha, e é de $n = 15$.

3 Problemas

3.1 Problema 1.1

Tabela 1: Maior n inteiro que pode ser resolvido em um tempo t, quando o problema demora $f(n)$ microssegundos

	1 s	1 m	1 h	1 d	1 mês	1 ano	1 séc.
$\lg n$	2^{10^6}	$2^{6.0\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.592\times10^{12}}$	$2^{3.1536\times10^{13}}$	$2^{3.1536\times10^{15}}$
$\sqrt{n}$	10^{12}	3.6×10^{15}	1.296×10^{19}	7.465×10^{21}	6.7185×10^{24}	9.9452×10^{26}	9.9452×10^{30}
n	10^6	6.0×10^7	3.6×10^9	8.64×10^{10}	2.592×10^{12}	3.1536×10^{13}	3.1536×10^{15}
$n \lg n$							
n^2	1000	7745	60000	293938	1609968	5615692	56175382
n^3	100	391	1532	4420	13736	31593	146677
2^n	19	25	31	36	41	44	51
$n!$							

Arrazoado:

- Sabendo que $f(n)$ demora $1\,\mu s$, então é fácil calcular o tempo para n: basta um cálculo proporcional direto. Por exemplo: em 1 segundo temos 1.000.000 de microssegundos, então o maior n que pode ser calculado em 1 segundo é 1.000.000 (10^6).
- Se a função demorar $\sqrt{n}$ microssegundos, basta calcular a função inversa, ou seja, n^2, para determinar qual o maior n.
- Quando a função demorar $\lg n$ (onde $\lg = \log_2$), o maior n é dado pela função inversa 2^n.
- Quando a função demorar n^2, o maior n é dado pela função inversa $\sqrt{n}$.
- Quando a função demorar n^3, o maior n é dado pela função inversa $\sqrt[3]{n}$.
- Quando a função demorar 2^n, o maior n é dado pela função inversa $\lg n$.
- $n \lg n$??
- $n!$??