Análise de Algoritmos

Introdução à Análise de Algoritmos

Profa. Sheila Morais de Almeida

DAINF-UTFPR-PG

março - 2018

Algoritmos

Definição

*"**Algoritmo** é a ideia por trás dos programas de computador. É aquilo que permanece igual se o programa estiver em Pascal rodando em um supercomputador da Cray em Nova Iorque ou se estiver em Basic rodando em um Mac em Catmandu! Um algoritmo resolve um problema especificado pelo conjunto de instâncias que devem ser tratadas e por quais as propriedades que a resposta deve ter."* Steven S. Skiena

Problema Computacional

Problemas Computacionais

Um **problema computacional** pode ser formalmente definido por uma função $f : E \rightarrow S$, onde E é conjunto das possíveis instâncias do problema e S é o conjunto das possíveis saídas para o problema. A função f mapeia cada instância do conjunto E na resposta correta para este caso, que é um elemento de S. Resolver o problema computacional $f : E \rightarrow S$ é criar um Algoritmo que, para cada elemento de E, responde corretamente apresentando o respectivo elemento em S.

Algoritmos

Exemplo: Problema da Ordenação

Entrada: uma sequência de números $a_1, a_2, a_3, \ldots, a_k$.

Saída: uma permutação $a'_1, a'_2, a'_3, \ldots a'_k$ da sequência original, de forma que $a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_k$.

No Problema da Ordenação, $f : E \rightarrow S$, E é o conjunto de ...

e S é o conjunto de ...

Algoritmos

Exemplo: Problema da Ordenação

Entrada: uma sequência de números $a_1, a_2, a_3, \ldots, a_k$.

Saída: uma permutação $a'_1, a'_2, a'_3, \ldots a'_k$ da sequência original, de forma que $a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_k$.

No Problema da Ordenação, $f : E \rightarrow S$, E é o conjunto de todas as sequências de números inteiros

e S é o conjunto de todas as sequências de números inteiros em ordem crescente.

Algoritmos

Exemplo: Problema da Ordenação

Fazer um algoritmo para o Problema da Ordenação, é criar um procedimento computacional que, para cada sequência s de E, apresenta a sequência de S dada pela $f(s)$.

Um algoritmo é **correto** se ele sempre para e sempre apresenta a resposta correta, ou seja, para cada sequência s, o algoritmo não pode apresentar qualquer sequência de S tem que ser $f(s)$.

Algoritmos

Formas de Representação

Algoritmos podem ser apresentados em língua natural, em linguagem de programação, como um projeto de hardware, dentre outras formas.

O importante é que a descrição seja suficientemente precisa para que possa ser reproduzida. Ou seja, a descrição não pode ser ambígua.

Análise de Algoritmos

Para um mesmo problema computacional podem existir muitos algoritmos diferentes. Conhecer seus pontos fortes e suas limitações pode ajudar a escolher qual deles é melhor para cada aplicação.

Análise de Algoritmos

Para analisar um algoritmo e poder compará-lo a outros que resolvem o mesmo problema, vamos avaliar sua **corretude** e **eficiência**.

Corretude:

Considerando qualquer instância válida, o algoritmo para? Para qualquer instância, o algoritmo apresenta a resposta esperada (de acordo com a especificação do problema)?

Eficiência

A eficiência do algoritmo é medida em termos da quantidade de recursos (memória, tempo de execução, número de processadores, acessos a disco) que o mesmo utiliza quando é executado.

Análise de Algoritmos

Em relação à eficiência, dois algoritmos desenvolvidos para resolver um mesmo problema podem ser drásticamente diferentes.

Essas diferenças podem ser muito mais significativas que diferenças de hardware ou software.

Análise de Algoritmos

Exemplo

Considere dois algoritmos que resolvem o Problema da Ordenação: *Insertion Sort* e *Merge Sort*.

Sabemos que o tempo de execução desses algoritmos varia conforme a quantidade de números que pecisam ser ordenados.

Veremos nessa disciplina que para ordenar uma sequência com n números, os tempos gastos são em média:

Insertion Sort: $c_1 n^2$, onde c_1 é uma constante que não depende de n.

Merge Sort: $c_2 n \log n$, onde c_2 é uma constante que não depende de n.

Análise de Algoritmos

Insertion Sort: $c_1 n^2$,

Merge Sort: $c_2 n \log n$,

Apesar de $c_1 < c_2$, o *Insertion Sort* só consegue ser melhor que o *Merge Sort* para instâncias pequenas (valor pequeno de n).

Quanto maior for n, maior é a diferença na eficiência entre esses algoritmos, destacando o desempenho do *Merge Sort*.

Interessante! Já que a constante no tempo de execução do *Insertion Sort* é menor.

Análise de Algoritmos

Suponha que o *Insertion Sort* (com tempo $c_1 n^2$) e o *Merge Sort* (com tempo $c_2 n \log n$) vão ser usados para ordenar um vetor com 1 milhão de elementos.

Mas vamos dar alguma vantagem para o *Insertion Sort*:

- computador do *Insertion Sort*: 1 bilhão de instruções por segundo;
- computador do *Merge Sort*: 10 milhões de instruções por segundo.

Ou seja, o *Insertion Sort* vai rodar em um computador 100 vezes mais rápido que o do *Merge Sort*!

Análise de Algoritmos

Vamos ordenar 1 milhão e elementos e vamos dar alguma vantagem para o *Insertion Sort*:

- o programador do *Insertion Sort* é o mais habilidoso do mundo e implementou em linguagem de máquina.

  Tempo de execução do *Insertion Sort*: $2n^2$.

- o programador do *Merge Sort* é mediano, e usou linguagem de alto nível com um compilador ineficiente.

  Tempo de execução do *Merge Sort*: $50n \log n$.

Análise de Algoritmos

Vamos ordenar 1 milhão e elementos e vamos dar alguma vantagem para o *Insertion Sort*:

- computador do *Insertion Sort*: 1 bilhão de instruções por segundo;
- computador do *Merge Sort*: 10 milhões de instruções por segundo.
- tempo de execução do *Insertion Sort*: $2n^2$.
- tempo de execução do *Merge Sort*: $50n \log n$.

Quem vai ser mais rápido?

Análise de Algoritmos

Contas do Cormen:

Insertion Sort: $\frac{2(10^6)^2 \text{ instruções}}{10^9 \text{ instruções por segundo}} = 2000 \text{ segundos}$

Merge Sort: $\frac{50(10^6 \log(10^6)) \text{ instruções}}{10^7 \text{ instruções por segundo}} = 100 \text{ segundos}$

Análise de Algoritmos

Vamos comparar algumas funções comuns que representam número de instruções:

	$n = 100$	$n = 1000$	$n = 10^4$	$n = 10^6$	$n = 10^9$
$\log n$	2	3	4	6	9
n	100	1000	10^4	10^6	10^9
$n \log n$	200	3000	$4 \cdot 10^4$	$6 \cdot 10^6$	$9 \cdot 10^9$
n^2	10^4	10^6	10^8	10^{12}	10^{18}
$100n^2 + 15n$	$1,0015 \cdot 10^6$	$1,00015 \cdot 10^8$	$\approx 10^{10}$	$\approx 10^{14}$	$\approx 10^{20}$
2^n	$\approx 1,26 \cdot 10^{30}$	$\approx 1,07 \cdot 10^{301}$	?	?	?

Análise de Algoritmos

Vamos comparar algumas funções comuns que representam número de instruções:

1 milhão (10^6) de operações por segundo

Função de custo	10	20	30	40	50	60
n	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s	0,00006s
n^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0025s	0,0036s
n^3	0,001s	0,008s	0,027s	0,064s	0,125s	0,216s
n^5	0,1s	3,2s	24,3s	1,7min	5,2min	12,96min
2^n	0,001s	1,04s	17,9min	12,7dias	35,7 anos	366 séc.
3^n	0,059s	58min	6,5anos	3855séc.	10^8séc.	10^{13}séc.

Análise de Algoritmos

A tabela abaixo, extraída do livro "The design and analysis of computer algorithms", de Aho, Hopcroft e Ullman, compara diferentes algoritmos considerando o mesmo período de tempo para a execução e o tamanho das instâncias que podem ser tratadas nesse prazo.

Algorithm	Time complexity	Maximum problem size		
		1 sec	1 min	1 hour
A_1	n	1000	6×10^4	3.6×10^6
A_2	$n \log n$	140	4893	2.0×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

Fig. 1.1. Limits on problem size as determined by growth rate.

Análise de Algoritmos

Algorithm	Time complexity	Maximum problem size		
		1 sec	1 min	1 hour
A_1	n	1000	6×10^4	3.6×10^6
A_2	$n \log n$	140	4893	2.0×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

Fig. 1.1. Limits on problem size as determined by growth rate.

Observe que em 1 minuto o algoritmo A_1 consegue resolver o problema para uma instância de tamanho 60.000, enquanto o algoritmo A_5 resolve o problema para uma instância de tamanho 15.

Faça outras comparações!

Análise de Algoritmos

Pergunta: Mas e se a gente tiver computadores muito mais potentes?

Não vamos mais precisar nos preocupar com a eficiência dos algoritmos?

Análise de Algoritmos

Suponha que os computadores se tornem 10 vezes mais rápidos que os da geração utilizada para as comparações da tabela anterior.

No livro de Aho, Hopcroft e Hullman, eles apresentam a seguinte tabela de comparação:

Algorithm	Time complexity	Maximum problem size before speed-up	Maximum problem size after speed-up
A_1	n	s_1	$10s_1$
A_2	$n \log n$	s_2	Approximately $10s_2$ for large s_2
A_3	n^2	s_3	$3.16s_3$
A_4	n^3	s_4	$2.15s_4$
A_5	2^n	s_5	$s_5 + 3.3$

Fig. 1.2. Effect of tenfold speed-up.

Análise de Algoritmos

Algorithm	Time complexity	Maximum problem size before speed-up	Maximum problem size after speed-up
A_1	n	s_1	$10s_1$
A_2	$n \log n$	s_2	Approximately $10s_2$ for large s_2
A_3	n^2	s_3	$3.16s_3$
A_4	n^3	s_4	$2.15s_4$
A_5	2^n	s_5	$s_5 + 3.3$

Fig. 1.2. Effect of tenfold speed-up.

Observe que no algoritmo A_4 um aumento de 10 vezes na velocidade de computação das instruções causa um aumento de pouco mais que 2 vezes no tamanho das entradas que se pode computar.

Análise de Algoritmos

Algorithm	Time complexity	Maximum problem size before speed-up	Maximum problem size after speed-up
A_1	n	s_1	$10s_1$
A_2	$n \log n$	s_2	Approximately $10s_2$ for large s_2
A_3	n^2	s_3	$3.16s_3$
A_4	n^3	s_4	$2.15s_4$
A_5	2^n	s_5	$s_5 + 3.3$

Fig. 1.2. Effect of tenfold speed-up.

No algoritmo A_5, o aumento de 10 vezes na velocidade de computação aumentou em menos de 4 **unidades** (não é vezes!) o tamanho das entradas que se pode computar.

Análise de Algoritmos

Você pode pensar: mas na minha área eu não preciso me preocupar com algoritmos!

Análise de Algoritmos

Algoritmos formam o núcleo da maioria das tecnologias usadas em algoritmos contemporâneos.

Considere a implementação de um serviço web que determina a melhor rota para ir de um lugar a outro.

O que é necessário para sua implementação?

- hardware rápido;
- interface gráfica com o usuário;
- redes remotas;
- (talvez) orientação a objeto.

Fonte: Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. Introduction to Algorithms, 2nd ed., 2001.

Análise de Algoritmos

Considere a implementação de um serviço web que determina a melhor rota para ir de um lugar a outro.

O que é necessário para sua implementação?

Também exigiria algoritmos para certas operações, como:

- encontrar rotas (provavelmente usando um algoritmo de caminho mais curto);
- interpretar mapas;
- interpolar endereços.

Fonte: Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. Introduction to Algorithms, 2nd ed., 2001.

Análise de Algoritmos

Outros casos em que algoritmos aparecem menos explícitos:

- aplicações que dependem de hardware de alto desempenho: o projeto de hardware usa algoritmos.
- aplicações que usam interface gráfica com o usuário: o projeto de qualquer interface gráfica com o usuário necessita de algoritmos.

Fonte: Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. Introduction to Algorithms, 2nd ed., 2001.

Análise de Algoritmos

Outros casos em que algoritmos aparecem menos explícitos:

- aplicações que dependem de redes de computadores: o roteamento em redes é fortemente dependente de algoritmos.
- aplicações desenvolvidas em outras linguagens que não código de máquina: seu código precisa ser processado por um compilador, interpretador ou assembler, os quais fazem uso extensivo de algoritmos.

Fonte: Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. Introduction to Algorithms, 2nd ed., 2001.

Análise de Algoritmos

Conclusão: Se você quer ser um programador realmente habilidoso, pecisa ter uma base sólida sobre projeto e análise de algoritmos!

Exercícios

Faça os exercícios das seções 1.1 e 1.2 do livro

Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. Introduction to Algorithms, 2nd ed., 2001.

Referências

Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. *Introduction to Algorithms*, 2 ed., 2001.

Alfred AHO, Jeffrey HOPCROFT, John ULLMAN. *The design and analysis of computer algorithms*, 1 ed., 1974.