Análise de Algoritmos

Primeira Lista de Exercícios

Fundamentos matemáticos da análise de algoritmos e Algoritmos de Ordenação

26 de março de 2008

Prof. Mauricio Ayala-Rincón

Entrega: 16 de abril de 2008 (100%) - Grupos de cinco pessôas
Será disponibilizado um gabarito da lista, depois do dia 16 de abril.
Listas entregues entre 17 e 23 de abril de 2008: 50%
Listas entregues após 23 de abril de 2008: 0%

Estágiario de docência: Daniel Lima Ventura: `ventura at mat dot unb dot br`

Fundamentos

1. Complete a seguinte tabela determinando o maior tamanho de um problema que pode ser solucionado em tempo t por um algoritmo $\mathcal{A}$, para cada função $f_{\mathcal{A}}(n)$ na seguinte tabela. Supor que o algoritmo $\mathcal{A}$ resolve um problema de tamanho n em tempo $f_{\mathcal{A}}(n)$ microsegundos.

$f_{\mathcal{A}}(n)$	1 seg.	1 min. $\times 60$	1 hora $\times 60$	1 dia $\times 24$	1 mês $\times 30$	1 ano $\times 12$	1 século $\times 100$
$ln(n)$							
$\sqrt{n}$							
n	10^6	$6\ 10^7$	$3.6\ 10^9$	$8.64\ 10^{10}$	$2.592\ 10^{12}$	$3.1104\ 10^{13}$	$3.1104\ 10^{15}$
$nln(n)$							
n^2	10^3	$\sqrt{60}\ 10^3$	$6\ 10^4$	$\sqrt{8.64}\ 10^5$	$\sqrt{2.592}\ 10^6$	$\sqrt{31.104}\ 10^6$	$\sqrt{31.104}\ 10^7$
n^3	10^2						
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Caso existesem computadores que executaram 1 instrução por nanosegundo, determine o maior tamanho de um problema que pode ser tratado em tempo t por algoritmos $\mathcal{A}$ com funções correspondentes $f_{\mathcal{A}}(n)$ na seguinte tabela.

$f_{\mathcal{A}}(n)$	1 seg.	1 min. $\times 60$	1 hora $\times 60$	1 dia $\times 24$	1 mês $\times 30$	1 ano $\times 12$	1 século $\times 100$
n	10^9	$6\ 10^{10}$	$3.6\ 10^{12}$	$8.64\ 10^{13}$	$2.592\ 10^{13}$	$3.1104\ 10^{16}$	$3.1104\ 10^{18}$
n^2							
n^3							
2^n							
$n!$							

2. Resolva as seguintes relações de recurrência:

   (a) $T(1) = 1$, $T(n) = 3T(n/2) + n^2$, $n \geq 2$;

   (b) $T(1) = 1$, $T(n) = 2T(n-1) + 1$, $n \geq 2$;

   (c) $T(1) = 1$, $T(n) = 2T(n/2) + n$, $n \geq 2$;

   (d) $T(1) \in \Theta(1)$, $T(n) = 3T(n/2) + nln(n)$;

   (e) $T(1) \in \Theta(1)$, $T(n) = 3T(n/3 + 5) + n/2$;

   (f) $T(1) \in \Theta(1)$, $T(n) = 2T(n/2) + n/ln(n)$;

   (g) $T(1) \in \Theta(1)$, $T(n) = T(n-1) + 1/n$;

   (h) $T(1) \in \Theta(1)$, $T(n) = T(n-1) + ln(n)$;

   (i) $T(1) \in \Theta(1)$, $T(n) = \sqrt{n}T(\sqrt{n}) + n$;

3. A função de Fibonacci é definida pela relação de recurrência

$$\begin{aligned} &F(0) = 0 \\ &F(1) = 1 \\ &F(n) = F(n-1) + F(n-2) \end{aligned}$$

   Seja $\mathcal{F}$, uma *função generatriz*, definida por

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} F(i) z^i$$

   Siga os seguintes passos para resolver a relação de recurrência da função de Fibonacci.

   (a) Demonstre que

$$\mathcal{F}(z) = \frac{z}{1 - z - z^2} = \frac{z}{(1 - \phi z)(1 - \tilde{\phi} z)} = \frac{1}{\sqrt{5}}\left(\frac{1}{1 - \phi z} - \frac{1}{1 - \tilde{\phi} z}\right),$$

   onde

$$\phi = \frac{1 + \sqrt{5}}{2} \quad \text{e} \quad \tilde{\phi} = \frac{1 - \sqrt{5}}{2}$$

   (b) Demonstre que

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}} (\phi^i - \tilde{\phi}^i) z^i$$

   (c) Prove que $F(i) = \phi^i / \sqrt{5}$ para $i > 0$, aproximado ao inteiro mais próximo.

   (d) Demonstre que $F(i + 2) \geq \phi^i$, para $i \geq 0$.

   (e) Conclua que $F(n) \in \Theta(\phi^n)$; i.e., a função de Fibonacci é de complexidade exponêncial.

Observe que também é possível estimar limites inferior e superior para a complexidade de $F(n)$ da seguinte maneira:

$$\forall n > 2, F(n) = F(n-1) + F(n-2) \quad \Rightarrow \quad 2\,F(n-2) \leq F(n) \leq 2\,F(n-1)$$

Considerando os casos n par e ímpar obtem-se:
n par:

$$\sqrt{2}^n/2 = 2^{n/2-1} = 2^{n/2-1}F(2) \leq F(n) \leq 2^{n-1}F(1) = 2^{n-1}$$

n ímpar:

$$\sqrt{2}^{n-1} = 2^{\lfloor n/2 \rfloor}F(1) \leq F(n) \leq 2^{n-1}F(1) = 2^{n-1}$$

Consequentemente, $F(n) \in O(2^n)$ e $F(n) \in \Omega(\sqrt{2}^n)$. Conclui-se que $F(n)$ é limitada inferior- e superiormente por funciones de classe de complexidade exponêncial.

Note que a conclusão inicial, $F(n) \in \Theta(\phi^n)$, é coerente com os limites anteriores, sempre que

$$\sqrt{2} < \phi = \frac{1+\sqrt{5}}{2} < 2$$

4. O problema das torres de Hanoi consiste em transladar n discos de diâmetros diferentes de uma torre A a uma torre C usando uma torre auxiliar B. Os discos estão inicialmente organizados na torre, A, decrescentemente, segundo o diâmetro e devem terminar na torre C com a mesma organização. Os discos são movidos de uma torre a outra com a restrição de que jamais se têm discos de diâmetro maior acima de discos de diâmetro menor numa mesma torre. Ademais, em um movimento não podem ser transladados varios discos simultaneamente e somente é permitido mover discos no tope das torres.

   A solução típica do problema, veja figura 1, consiste em:

   - transladar recursivamente os $n-1$ discos de menor diâmetro, da torre A à torre B, usando como torre auxiliar a torre C;
   - mover o disco de maior diâmetro de A a C;
   - Transladar recursivamente os $n-1$ discos de B a C, usando como torre auxiliar A.

   Seja $M(n)$ o número de movimentos necessários para transladar n discos segundo o algoritmo anterior.

   (a) Explique porque a seguinte relação de recurrência expressa o número correto de movimentos:

   $$M(n) = \begin{cases} 1, \text{ caso } n = 1 \\ 2\,M(n-1) + 1, \text{ caso } n > 1 \end{cases}$$

   (b) Calcule $M(n)$.

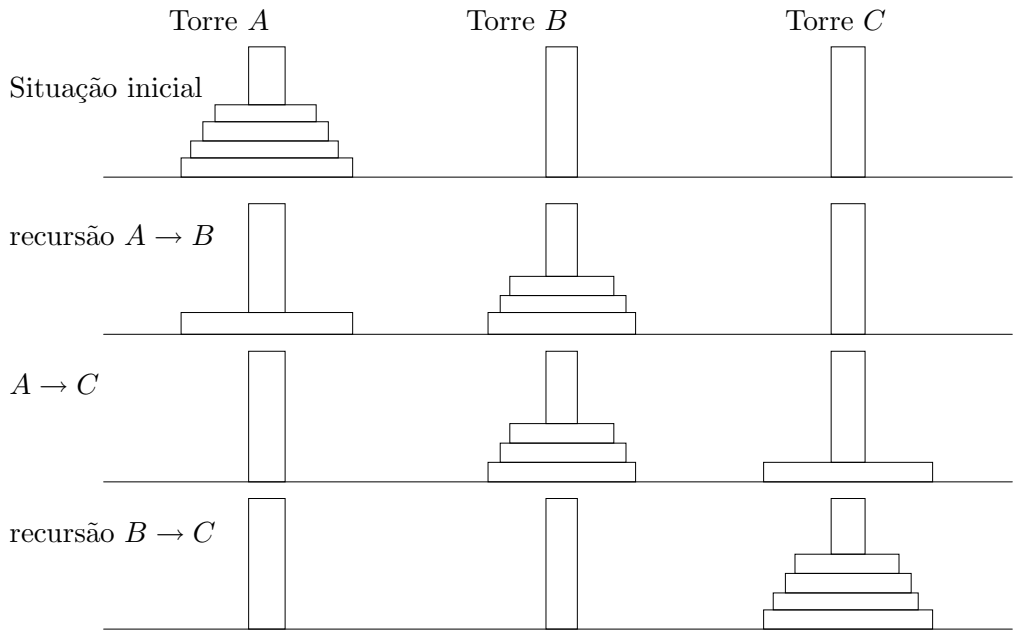

Figura 1: Torres de Hanoi

Algoritmos de Ordenação

Implementações na linguagem C dos algoritmos *maxsort, bubblesort, insertionsort, binary-insertionsort, quicksort, mergesort* e *heapsort* podem-se achar na página `http://www.mat.unb.br/~ayala/LECTURES/AA/AA.html` no arquivo "sorting.c".

5. Analise a complexidade, número de comparações, do algoritmo *binary-insertionsort* (i.e., *insertionsort* modificado, de maneira que a inserção é realizada por meio de busca binária).

6. Demonstre formalmente que qualquer algoritmo de fusão de listas faz ao menos cinco comparações, para fusionar duas listas ordenadas de comprimentos quatro e dois.
   Note que o número de possíveis formas do *merge* de listas ordenadas a_1, a_2, a_3, a_4 e b_1, b_2 é quinze. Verifique isto observando que cada uma das chaves b_1 e b_2 pode-se inserir na lista das a's de cinco formas diferentes, mas estas restritas mutuamente. Veja a Figura 2.

   Assim, segundo o limite inferior de otimização, poderíamos ter algoritmos de no mínimo $\lceil log_2(15) \rceil = 4$ comparações. Tem-se, então, que descrever um "pior caso" que precisa ao menos de cinco comparações.

7. Segundo o limite inferior de otimização para algoritmos de ordenação por comparação de chaves (i.e., $\lceil log_2(n!) \rceil$, onde n é o comprimento da lista) examinada na aula, uma lista de comprimento três ordena-se otimamente com $\lceil log_2(6) \rceil = 3$ comparações. Assim, que os algoritmos *maxsort*, *bubblesort*, *insertionsort*, *binary-insertionsort*, *quick-*

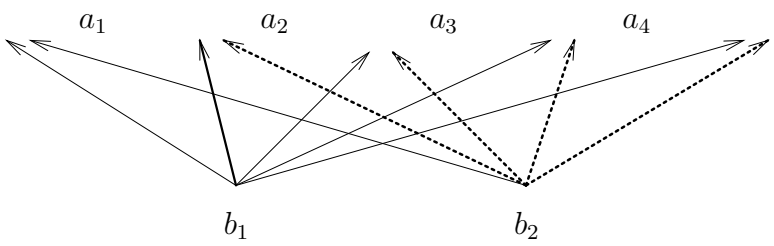

Figura 2: Fusão de listas ordenadas: inserção restrita

sort, *mergesort* e *heapsort* são ótimos para listas de comprimento três, pois no pior caso fazem três comparações (verifique!). Listas de comprimento quatro poderiam, então, ordenar-se com $\lceil log_2(24) \rceil = 5$ comparações. Note que, por exemplo, os algoritmos *maxsort*, *bubblesort*, *insertionsort* e *quicksort* fazem seis comparações (em tanto que o algoritmo *heapsort* faz sete comparações) no pior caso. Neste caso, os algoritmos *mergesort* e *binary-insertionsort* são ótimos (verifique!). Uma simples modificação do algoritmo *heapsort*[1] precisa seis comparações.
Agora, para listas de comprimento cinco temos um limite de otimização $\lceil log_2(120) \rceil = 7$. **Verifique** que os algoritmos *bubblesort*, *insertionsort* e *quicksort* fazem dez comparações (em tanto que o algoritmo *heapsort* faz doze comparações) e os algoritmos *mergesort* e *binary-insertionsort* oito comprações, no pior caso. Com uma simples modificação do algoritmo *heapsort* podem-se atingir dez comparações no pior caso.
Pode-se desenhar um algoritmo de ordenação para listas de comprimento cinco que precise no máximo de sete comparações?

8. No ano de 1974 (A. H. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addisson-Wesley, 1974) os melhores algoritmos conhecidos, baseados em comparação de chaves, para ordenar listas de comprimento n realizabam um número de comparações segundo a tabela embaixo.

comprimento	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
comparações	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50

Utilizando os recursos e métodos computacionais atualmente disponíveis a tabela poderia ser aumentada. Esquematize métodos algorítmicos que baseados estrutura das possíveis *árvores de decisão* associadas com algoritmos de ordenação para um número fixo de chaves, descrevam os melhores algoritmos de ordenação.

Conhecimento de tais algoritmos é relevante para refinar os métodos gerais mais populares. P.ex., o *mergesort* pode ser aprimorado modificando o mecanismo de escape do processo recursivo de forma tal que sublistas de comprimento igual que 5 sejam ordenadas com somente sete comparações, como na tabela, e não com oito. Análise

[1]Para listas de comprimento quatro a construção do *heap* precisa quatro comparações e a ordenação só dois, se para o primeiro *fixheap* usa-se toda a informação disponível, i.e., a chave a inserir é menor que o filho esquerdo da estrutura do *heap*.

p.ex. as vantagens, quando utilizamos *mergesort* assim aprimorado para ordenar uma lista de tamanho 5×2^k.

9. Os algoritmos de ordenação examinados na aula são desenhados pensando no caso de listas sem repetições de chaves (ou com poucas repetições). Para listas com repetições nossos algoritmos são corretos, mas não aproveitam a ocorrência de repetições.

   (a) Descreva o trabalho no pior caso ($W(n)$) para os algoritmos *insertionsort*, *quicksort* e *mergesort* de listas de 0's e 1's.

   (b) Desenhe um algoritmo "eficiente" baseado em comparação e intercâmbio de chaves (não use contadores extra!) para ordenar listas de 0's e 1's. Calcule o $W(n)$ de seu algoritmo.

   (c) (*The Dutch National Flag Problem*). Cada uma das n chaves de uma lista pode ser *vermelha*, *branca* ou *azul*. Apresente um algoritmo "eficiente" para ordenar este tipo de listas segundo a ordem $vermelha < branca < azul$. As únicas operações permitidas são comparação e intercâmbio de chaves (não use contadores extra!). Existe uma solução $\Theta(n)$.

10. Explique em detalhe e análise a complexidade tempo espaço dos algoritmos de ordenação *Shellsort, Countingsort, Radixsort* e *Bucketsort* na bibliografia da disciplina.