MAC 5711 - Análise de Algoritmos

Rodrigo Augusto Dias Faria
Departamento de Ciência da Computação - IME/USP

13 de setembro de 2015

Demonstração. Blah, blah, blah. Here is an example of the `align` environment:
□

Disproof. Blah, blah, blah. I'm so smart. □

Lista 1

1. Lembre-se que lg n denota o logaritmo na base 2 de n. Usando a definição de notação O, prove que

(a) $3n$ não é $O(2^n)$
(b) $\log_{10} n$ é $O(\lg n)$
(c) $\lg n$ é $O(\log_{10} n)$

Lista 2

2. Escreva um algoritmo que ordena uma lista de n itens dividindo-a em três sublistas de aproximadamente $n/3$ itens, ordenando cada sublista recursivamente e intercalando as três sublistas ordenadas. Analise seu algoritmo concluindo qual é o seu consumo de tempo.

Para este exercício, devemos efetuar uma alteração no MERGESORT para a divisão do vetor A em três partições utilizando o MERGE duas vezes ao final para intercalar as três partes ordenadas em um único vetor.

MERGESORT3(A, p, r)

```
1  if p < r
2      k = ⌊(p + r)/3⌋
3      m = k + 1 + ⌊(p + r)/3⌋
4      MERGESORT3(A, p, k)
5      MERGESORT3(A, k + 1, m)
6      MERGESORT3(A, m + 1, r)
7      MERGE(A, p, k, m)
8      MERGE(A, p, m, r)
```

Consumo de tempo

As linhas 1-3 consomem $\Theta(1)$. As linhas 4-5 têm consumo $T(\lceil n/3 \rceil)$ e a linha 6 tem consumo $T(n - \lceil 2n/3 \rceil)$, já que a terceira partição não tem tamanho exatamente de $\lceil n/3 \rceil$. Sabemos que o consumo do MERGE é $\Theta(n)$, logo:

$$\begin{aligned} T(n) &= T(\lceil n/3 \rceil) + T(\lceil n/3 \rceil) + T(n - 2\lceil n/3 \rceil) + \Theta(n) + \Theta(n) \\ &= 2T(\lceil n/3 \rceil) + T(\lceil n/3 \rceil) + \Theta(2n) \\ &= 3T(\lceil n/3 \rceil) + \Theta(2n) \end{aligned}$$

Como $\Theta(2n)$ é $\Theta(n)$:

$$T(n) = 3T(\lceil n/3 \rceil) + \Theta(n)$$

Simplificando a recorrência, temos:

$$T(n) = \begin{cases} 1, & n = 1 \\ 3T\left(\dfrac{n}{3}\right) + n, & n >= 2 \text{ potência de } 2 \end{cases}$$

Por expansão:

$$
\begin{aligned}
T(n) &= 3T\left(\frac{n}{3}\right) + n \\
&= 3\left(3T\left(\frac{n}{3^2}\right) + \left(\frac{n}{3}\right)\right) + n & &= 3^2T\left(\frac{n}{3^2}\right) + n + n \\
&= 3^2\left(3T\left(\frac{n}{3^3}\right) + \left(\frac{n}{3^2}\right)\right) + n + n & &= 3^3T\left(\frac{n}{3^3}\right) + n + n + n \\
&= ... \\
&= 3^kT\left(\frac{n}{3^k}\right) + kn
\end{aligned}
$$

Assumindo $k = \log_3 n$ e $3^k = n$:

$$
\begin{aligned}
T(n) &= nT\left(\frac{n}{n}\right) + \log_3 n \\
&= T(1)n + \log_3 n(n) \\
&= n + n(\log_3 n)
\end{aligned}
$$

Portanto, $T(n) = n + n(\log_3 n)$ é $\Theta(n \log n)$.

Demonstração. Prova por indução em k.

Base: para $n = 1$

$$T(1) = 1 = 1 + 1(\log_3 1) = 1 + 0 = 1$$

Hipótese de Indução: Assuma que $T(x) = x + x(\log_3 x)$ vale para $1 >= x < n$

Passo: para $n >= 2$

$$
\begin{aligned}
T(n) = 3T\left(\frac{n}{3}\right) + n &= 3\left(\left(\frac{n}{3}\right) + \left(\frac{n(\log_3 \frac{n}{3})}{3}\right)\right) + n & &\text{(por HI)} \\
&= 3\left(\frac{n}{3}\right) + 3\left(\left(\frac{n}{3}\right)\log_3 \frac{n}{3}\right) + n \\
&= n + n + n\log_3 \frac{n}{3} \\
&= 2n + n\log_3 n - n \\
&= n + n\log_3 n
\end{aligned}
$$

Como queríamos demonstrar!

□

Lista 3

2. Qual é o consumo de espaço do QUICKSORT no pior caso?

A avaliação de um algoritmo quanto ao consumo de espaço está relacionada com a necessidade de alocação de espaço adicional na pilha de recursão.

No pior caso, o QUICKSORT será executado uma vez para cada elemento da lista dada de tamanho n, ou seja, teremos n chamadas recursivas.

Isso significa que, com uma lista de n elementos, n novas chamadas serão adicionadas à pilha no pior caso, o que nos leva a uma complexidade de espaço $\mathrm{O}(n)$.

Lista 4

1. Escreva uma função que recebe um vetor com n letras A's e B's e, por meio de trocas, move todos os A's para o início do vetor. Sua função deve consumir tempo O(n).

Resposta

3. Sejam $X[1..n]$ e $Y[1..n]$ dois vetores, cada um contendo n números ordenados. Escreva um algoritmo O(lg n) para encontrar uma das medianas de todos os $2n$ elementos nos vetores X e Y.

Sabemos que a mediana de X e Y está em $i = \lfloor q/2 \rfloor$ e $j = \lfloor s/2 \rfloor$, respectivamente. Note que $n = q + s$ é par, e é por isso que nós estamos usando a função **piso**.

Se $X[i]$ é maior do que $Y[j]$, significa que a mediana global está à esquerda de $X[i]$ e à direita de $Y[j]$. Se $X[i]$ é menor ou igual a $Y[j]$, nós procuramos a mediana à esquerda de $Y[j]$ e à direita de $X[i]$.

A condição de parada dá-se quando $p == q$, o que significa que a mediana global está dentro do vetor X. Caso contrário, se $r == s$, a mediana está em Y.

O pseudocódigo FIND-MEDIAN mostra a operação descrita acima que, também, é o resultado do exercício 9.3-8 CLRS 3ed.

```
FIND-MEDIAN(X, Y, p, q, r, s)
if p == q
// We have found the median between p, q and r
     return X[p]
elseif r == s
// We have found the median between q, r and s
     return Y[r]
i = p + (q − p)/2
j = r + (s − r)/2
if X[i] > Y[j]
     q = i
     r = j
else
     p = i
     s = j
return FIND-MEDIAN(X, Y, p, q, r, s)
```

4. (**CLRS 9.3-5**) Para esta questão, vamos dizer que a mediana de um vetor $A[p..r]$ com números inteiros é o valor que ficaria na posição $A[\lfloor (p + r)/2 \rfloor]$ depois que o vetor $A[p..r]$ fosse ordenado.

Dado um algoritmo linear "caixa-preta" que devolve a mediana de um vetor, descreva um algoritmo simples, linear, que, dado um vetor $A[p..r]$ de inteiros distintos e um inteiro k, devolve o *k-ésimo* mínimo do vetor. (O *k-ésimo* mínimo de um vetor de inteiros distintos é o elemento que estaria na *k-ésima* posição do vetor se ele fosse ordenado.)

Resposta

8. (**CLRS 8.3-2**) Quais dos seguintes algoritmos de ordenação são estáveis: insertionsort, mergesort, heapsort, e quicksort. Descreva uma maneira simples de deixar qualquer algoritmo de ordenação estável. Quanto tempo e/ou espaço adicional a sua estratégia usa?

Os algoritmos estáveis são o insertionsort e o mergesort (versão do Cormen). Os demais não são estáveis.

Uma forma simples de deixar qualquer algoritmo de ordenação estável é criar um mecanismo de indexação que mantenha a ordem em que os elementos aparecem originalmente, ou seja, basta termos um índice para cada elemento de um vetor de n elementos.

Esse mecanismo necessita de $\Theta(n)$ espaço extra para armazenar os n índices do vetor de n elementos.