Projeto e Análise de Algoritmos*

Introdução a algoritmos aleatorizados e ordenação por particionamento

Lehilton Pedrosa

Segundo Semestre de 2017

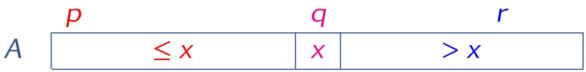
*Criado por C. de Souza, C. da Silva, O. Lee, F. Miyazawa et al.

Quick Sort

QuickSort

O algoritmo QuickSort segue o paradigma de divisão-e-conquista.

Divisão: divida o vetor em dois subvetores $A[p \ldots q-1]$ e $A[q+1 \ldots r]$ tais que



$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

Conquista: ordene os dois subvetores recursivamente usando o QuickSort;

Combinação: nada a fazer, o vetor está ordenado.

Problema: Rearranjar um dado vetor $A[p \dots r]$ e devolver um índice q, $p \leq q \leq r$, tais que

$$A[p \dots q-1] \leq A[q] < A[q+1 \dots r]$$

Entrada:

	p									r
A	99	33	55	77	11	22	88	66	33	44

Saída:

	p				q					r
A	33	11	22	33	44	55	99	66	77	88

Particione

	p									r
A	99	33	55	77	11	22	88	66	33	44

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

	i				j					x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

Particione

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Particione

Rearranja $A[p \dots r]$ de modo que $p \leq q \leq r$ e
$A[p \dots q{-}1] \leq A[q] < A[q{+}1 \dots r]$

Particione(A, p, r)
1 $\quad x \leftarrow A[r]$ $\quad \triangleright\, x$ é o "pivô"
2 $\quad i \leftarrow p{-}1$
3 $\quad$ **para** $j \leftarrow p$ **até** $r-1$ **faça**
4 $\quad\quad$ **se** $A[j] \leq x$
5 $\quad\quad\quad$ **então** $i \leftarrow i+1$
6 $\quad\quad\quad\quad\quad A[i] \leftrightarrow A[j]$
7 $\quad A[i{+}1] \leftrightarrow A[r]$
8 $\quad$ **devolva** $i+1$

Invariantes:

No começo de cada iteração da linha 3 vale que:
(1) $A[p \dots i] \leq x$ $\qquad$ (2) $A[i{+}1 \dots j{-}1] > x$ $\qquad$ (3) $A[r] = x$

Complexidade de PARTICIONE

PARTICIONE(A, p, r)	Tempo
$x \leftarrow A[r]$ $\triangleright$ x é o "pivô"	$\Theta(1)$
$i \leftarrow p-1$	$\Theta(1)$
para $j \leftarrow p$ **até** $r-1$ **faça**	$\Theta(n)$
se $A[j] \leq x$	$\Theta(n)$
então $i \leftarrow i+1$	$O(n)$
$A[i] \leftrightarrow A[j]$	$O(n)$
$A[i+1] \leftrightarrow A[r]$	$\Theta(1)$
devolva $i+1$	$\Theta(1)$

Se $n := r - p + 1$, então a complexidade no pior caso é

$$T(n) = \Theta(2n+4) + O(2n) = \Theta(n)$$

QuickSort

Rearranja um vetor $A[p \dots r]$ em ordem crescente.

QuickSort(A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ Particione(A, p, r)
3 QuickSort$(A, p, q-1)$
4 QuickSort$(A, q+1, r)$

	p									r
A	99	33	55	77	11	22	88	66	33	44

QuickSort

Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

QuickSort(A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ Particione(A, p, r)
3 QuickSort$(A, p, q-1)$
4 QuickSort$(A, q+1, r)$

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

No começo da linha 3, vale

$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

QuickSort

Rearranja um vetor $A[p \dots r]$ em ordem crescente.

```
QUICKSORT(A, p, r)
    se p < r
        então q ← PARTICIONE(A, p, r)
              QUICKSORT(A, p, q − 1)
              QUICKSORT(A, q + 1, r)
```

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

QuickSort

Rearranja um vetor $A[p \dots r]$ em ordem crescente.

$\textsc{QuickSort}(A, p, r)$
1 **se** $p < r$
2 **então** $q \leftarrow \textsc{Particione}(A, p, r)$
3 $\textsc{QuickSort}(A, p, q-1)$
4 $\textsc{QuickSort}(A, q+1, r)$

	p				q					r
A	11	22	33	33	44	55	66	77	88	99

Complexidade de QuickSort

QuickSort(A, p, r)	Tempo
1 **se** $p < r$	$\Theta(1)$
2 **então** $q \leftarrow$ Particione(A, p, r)	$\Theta(n)$
3 QuickSort$(A, p, q-1)$	$T(k)$
4 QuickSort$(A, q+1, r)$	$T(n-k-1)$

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n-1$.

$$T(n) = T(k) + T(n-k-1) + \Theta(n+1)$$

Recorrência

$T(n) :=$ consumo de tempo no pior caso

$$
\begin{aligned}
T(0) &= \Theta(1) \\
T(1) &= \Theta(1) \\
T(n) &= T(k) + T(n-k-1) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

Recorrência de um caso (vetor ordenado):

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

$T(n)$ é $\Theta(n^2)$.

Recorrência para o pior caso

$T(n) :=$ complexidade de tempo no pior caso

$$
\begin{aligned}
T(0) &= \Theta(1) \\
T(1) &= \Theta(1) \\
T(n) &= \max_{0 \le k \le n-1} \{T(k) + T(n-k-1)\} + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

$$T(n) = \max_{0 \le k \le n-1} \{T(k) + T(n-k-1)\} + bn$$

Quero mostrar que $T(n) = \Theta(n^2)$.

Demonstração: $T(n) = O(n^2)$

Vou mostrar $T(n) \leq cn^2$ por indução em n (grande).

$$\begin{aligned}
T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + bn \\
&\leq \max_{0 \leq k \leq n-1} \left\{ ck^2 + c(n-k-1)^2 \right\} + bn \\
&= c \max_{0 \leq k \leq n-1} \left\{ k^2 + (n-k-1)^2 \right\} + bn \\
&= c(n-1)^2 + bn \qquad \rhd \text{ exercício} \\
&= cn^2 - 2cn + c + bn \\
&\leq cn^2,
\end{aligned}$$

se $c > b/2$ e $n \geq c/(2c-b)$.

Continuação: $T(n) = \Omega(n^2)$

Agora vou mostrar que $T(n) \geq dn^2$ para n grande.

$$\begin{aligned}
T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + bn \\
&\geq \max_{0 \leq k \leq n-1} \left\{ dk^2 + d(n-k-1)^2 \right\} + bn \\
&= d \max_{0 \leq k \leq n-1} \left\{ k^2 + (n-k-1)^2 \right\} + bn \\
&= d(n-1)^2 + bn \\
&= dn^2 - 2dn + d + bn \\
&\geq dn^2,
\end{aligned}$$

se $d < b/2$ e $n \geq d/(2d-b)$.

Conclusão

$T(n)$ é $\Theta(n^2)$.

A complexidade de tempo do QuickSort no pior caso é $\Theta(n^2)$.

QuickSort no melhor caso

$M(n) :=$ complexidade de tempo no melhor caso

$$M(0) = \Theta(1)$$
$$M(1) = \Theta(1)$$
$$M(n) = \min_{0 \le k \le n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Mostre que, para $n \ge 1$,

$$M(n) \ge \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

Isto implica que no melhor caso o QuickSort é $\Omega(n \lg n)$.

QuickSort no melhor caso

No melhor caso k é aproximadamente $(n-1)/2$.

$$R(n) = R\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + R\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$.

Humm, lembra a recorrência do MERGE-SORT...

Mais algumas conclusões

$M(n)$ é $\Theta(n \lg n)$.

O consumo de tempo do QUICKSORT no melhor caso é $\Omega(n \log n)$.

Caso médio

Apesar da complexidade de tempo do QuickSort no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QuickSort no caso médio é mais próximo do melhor caso do que do pior caso.

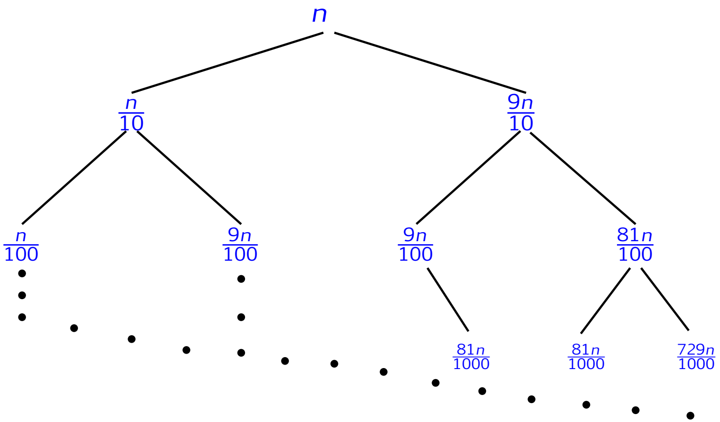
Por quê??

Suponha que (por sorte) o algoritmo Particione sempre divide o vetor na proporção $\frac{1}{9}$ para $\frac{9}{10}$. Então

$$T(n) = T\left(\left\lfloor \frac{n-1}{9} \right\rfloor\right) + T\left(\left\lceil \frac{9(n-1)}{10} \right\rceil\right) + \Theta(n)$$

Solução: $T(n)$ é $\Theta(n \lg n)$.

Árvore de recorrência



Número de níveis $\leq \log_{10/9} n$.

Em cada nível o custo é $\leq n$.

Custo total é $O(n \log n)$.

QuickSort Aleatório

O pior caso do QuickSort ocorre devido a uma escolha infeliz do pivô.

Um modo de minimizar este problema é usar aleatoriedade.

```
PARTICIONE-ALEATÓRIO(A, p, r)
    i ← RANDOM(p, r)
    A[i] ↔ A[r]
    devolva PARTICIONE(A, p, r)
```

```
QUICKSORT-ALEATÓRIO(A, p, r)
    se p < r
       então q ← PARTICIONE-ALEATÓRIO(A, p, r)
             QUICKSORT-ALEATÓRIO(A, p, q − 1)
             QUICKSORT-ALEATÓRIO(A, q + 1, r)
```

Tempo Esperado do QuickSort

O número esperado de comparações do *QuickSort* é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são distintos. Sejam $z_1 < z_2 < \ldots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ e X_{ij} variável aleatória que indica que z_i foi comparado com z_j:

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o número total de comparações X é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

Pela linearidade da esperança:

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

Como X_{ij} é uma variável aleatória binária,

$$\begin{aligned} E[X_{ij}] &= 0 \cdot \Pr\{z_i \text{ não ser comparado com } z_j\} + \\ &\quad 1 \cdot \Pr\{z_i \text{ ser comparado com } z_j\} \\ &= \Pr\{z_i \text{ ser comparado com } z_j\} \end{aligned}$$

Tempo Esperado do QuickSort

Considere a escolha do pivô e a comparação entre z_i e z_j:

$$\underbrace{z_1, z_2, \ldots, z_{i-1}}_{\text{posterga}}, \mathbf{z_i}, \underbrace{z_{i+1}, \ldots, z_{j-1}}_{\text{não comp.}}, \mathbf{z_j}, \underbrace{z_{j+1}, \ldots, z_n}_{\text{posterga}}$$

Assim,

$$\begin{aligned}
&\Pr\{z_i \text{ ser comparado com } z_j\} \\
&\quad = \Pr\{z_i \text{ ou } z_j \text{ é escolhido como pivô primeiro em } Z_{ij}\} \\
&\quad = \Pr\{z_i \text{ é escolhido como pivô primeiro em } Z_{ij}\} + \\
&\quad\ \Pr\{z_j \text{ é escolhido como pivô primeiro em } Z_{ij}\} \\
&\quad = \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&\quad = \frac{2}{j-i+1}.
\end{aligned}$$

Tempo Esperado do QuickSort

Portanto,

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n)
\end{aligned}
$$

Conclusão

O consumo de tempo esperado pelo QUICKSORT-ALEATÓRIO para itens distintos é $O(n \lg n)$.

Exercício Mostre que $T(n) = \Omega(n \lg n)$.

Conclusão:

O consumo de tempo esperado pelo QUICKSORT-ALEATÓRIO para itens distintos é $\Theta(n \lg n)$.

Conclusão

Exercício Analise o tempo esperado do QuickSort-Aleatório quando todos os elementos são iguais.

Exercício Adapte o algoritmo QuickSort-Aleatório para executar em tempo esperado $O(n \lg n)$ quando há elementos iguais.
Dica: Faça uma variante do Particione-Aleatório que considera elementos iguais, dividindo o vetor em três partes, de acordo com o pivô (partes com elementos menores, iguais e maiores que o pivô).