Quicksort

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Conteúdo

- Descrição do algoritmo QUICKSORT.
- Correção.
- Análise de desempenho.
- Versão aleatória.
- Exercícios.

Introdução

Quicksort

- Algoritmo de ordenação proposto por C.A.R. Hoare (1960).[1]
- Estudaremos uma versão desenvolvida por N. Lomuto.[2]
- Baseado no paradigma de divisão e conquista.
- Bom desempenho na prática.
- Comparação com o Merge-sort:

	Quicksort	Merge-sort
Tempo de execução no pior caso	$\Theta(n^2)$	$\Theta(n \lg n)$
Tempo de execução esperado	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Ordenação local	Sim	Não
Estável	Não	Sim

[1] Ver versão original em ▸ Sedgewick-Wayne.

[2] Versão apresentada no livro do Cormen.

Descrição do QUICKSORT

Divisão e conquista

Para ordenar um vetor $A[p \ldots r]$:

- **Dividir**: divida o vetor $A[p \ldots r]$ em dois subvetores $A[p \ldots q{-}1]$ e $A[q{+}1 \ldots r]$, tal que $A[p \ldots q{-}1] \leq A[q] < A[q{+}1 \ldots r]$.
- **Conquistar**: Ordenar os dois subvetores *recursivamente* usando o QUICKSORT.
- **Combinar**: Como os subvetores são ordenados localmente, não é necessário nenhum trabalho para combiná-los.

Descrição do QUICKSORT

Divisão e conquista

Para ordenar um vetor $A[p \ldots r]$:

- **Dividir**: divida o vetor $A[p \ldots r]$ em dois subvetores $A[p \ldots q{-}1]$ e $A[q{+}1 \ldots r]$, tal que $A[p \ldots q{-}1] \leq A[q] < A[q{+}1 \ldots r]$.
- **Conquistar**: Ordenar os dois subvetores *recursivamente* usando o QUICKSORT.
- **Combinar**: Como os subvetores são ordenados localmente, não é necessário nenhum trabalho para combiná-los.

Importante

O passo de divisão é feito pelo procedimento PARTITION, que devolve o índice q que marca a posição de divisão dos subvetores.

Procedimento Quicksort

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

$\textsc{Quicksort}(A, p, r)$

```
1 if p < r then
2 |   q ← Partition(A, p, r)
3 |   Quicksort(A, p, q − 1)
4 |   Quicksort(A, q + 1, r)
```

- Para ordenar todo um array A, a chamada inicial é $\textsc{Quicksort}(A, 1, A.length)$.
- Antes de entender o Quicksort, temos que entender o Partition.

O que faz o procedimento PARTITION?

Problema

Rearranjar $A[p \ldots r]$ e devolver um índice q, $p \leq q \leq r$, tal que:

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

Entrada

$$A = [8, 1, 6, 4, 0, 3, 9, 5]$$

Saída

$$A = [1, 4, 0, 3, 5, 8, 9, 6]$$

q = 5 (índice).

Procedimento PARTITION

```
PARTITION(A, p, r)
1 x ← A[r]                                   // x é o pivô
2 i ← p − 1
3 for j ← p to r − 1 do
4 |   if A[j] ≤ x then
5 |   |   i ← i + 1
6 |   |   SWAP(A[i], A[j])
7 SWAP(A[i + 1], A[r])
8 return i + 1
```

Invariantes

No começo de cada iteração da linha 3:

1. $A[p \ldots i] \leq x$.
2. $A[i{+}1 \ldots j{-}1] > x$.
3. $A[r] = x$.

Correção do Partition

Demonstração

- **Inicialização**: Antes do início do laço todas as condições da invariante são satisfeitas, porque o r é o pivô e os subvetores $A[p \ldots i]$ e $A[i{+}1 \ldots j{-}1]$ são vazios.
- **Manutenção**: Dois casos:
  - $A[j] > x$: j é incrementado e o invariante 2 se mantém para a próxima iteração. Como o índice i não foi alterado, o invariante 1 também se mantém.
  - $A[j] \leq x$: i é incrementado e na posição $A[i{+}1]$ é colocado o valor de $A[j]$ (mantendo o invariante 1). Sabe-se o valor do elemento na posição $i + 1$ era maior que o pivô, portanto, ao incrementar o valor de j, o invariante 2 se mantém.
- **Término**: Quando o laço termina, $j = r$, todos os elementos de A estão particionados em um dos três casos: $A[p \ldots i] \leq x$, $A[i{+}1 \ldots j{-}1] > x$ e $A[r] = x$.

Desempenho do PARTITION

Complexidade de tempo ($n = r - p + 1$)

PARTITION(A, p, r)	Tempo
1 $x \leftarrow A[r]$ $\quad //x$ é o pivo	
2 $i \leftarrow p - 1$	
3 **for** $j \leftarrow p$ **to** $r - 1$ **do**	
4 $\quad$ **if** $A[j] \leq x$ **then**	
5 $\quad\quad i \leftarrow i + 1$	
6 $\quad\quad$ SWAP$(A[i], A[j])$	
7 SWAP$(A[i + 1], A[r])$	
8 **return** $i + 1$	

Desempenho do PARTITION

Complexidade de tempo ($n = r - p + 1$)

PARTITION(A, p, r)	Tempo
1 $x \leftarrow A[r]$ $\quad //x$ é o pivo	$\Theta(1)$
2 $i \leftarrow p - 1$	$\Theta(1)$
3 **for** $j \leftarrow p$ **to** $r - 1$ **do**	$\Theta(n)$
4 **if** $A[j] \leq x$ **then**	$\Theta(n)$
5 $i \leftarrow i + 1$	$O(n)$
6 SWAP$(A[i], A[j])$	$O(n)$
7 SWAP$(A[i + 1], A[r])$	$\Theta(1)$
8 **return** $i + 1$	$\Theta(1)$

Conclusão

A complexidade de PARTITION é $\Theta(n)$.

Procedimento Quicksort

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

$\textsc{Quicksort}(A, p, r)$

1 **if** $p < r$ **then**
2 $\quad q \leftarrow \textsc{Partition}(A, p, r)$
3 $\quad \textsc{Quicksort}(A, p, q-1)$
4 $\quad \textsc{Quicksort}(A, q+1, r)$

Considere a entrada:

$$A = [8, 1, 6, 4, 0, 3, 9, 5]$$

Procedimento Quicksort

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

$\textsc{Quicksort}(A, p, r)$

1 **if** $p < r$ **then**
2 $\quad q \leftarrow \textsc{Partition}(A, p, r)$
3 $\quad \textsc{Quicksort}(A, p, q-1)$
4 $\quad \textsc{Quicksort}(A, q+1, r)$

Considere a entrada:

$$A = [8, 1, 6, 4, 0, 3, 9, 5]$$

Após a execução da linha 2 temos:

$$A = [1, 4, 0, 3, \mathbf{5}, 8, 9, 6]$$

Procedimento Quicksort

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

$\textsc{Quicksort}(A, p, r)$

1 **if** $p < r$ **then**
2 $\quad q \leftarrow \textsc{Partition}(A, p, r)$
3 $\quad \textsc{Quicksort}(A, p, q-1)$
4 $\quad \textsc{Quicksort}(A, q+1, r)$

Considere a entrada:

$$A = [8, 1, 6, 4, 0, 3, 9, 5]$$

Chamada recursiva da linha 3 para a parte em vermelho:

$$A = [\mathbf{1}, \mathbf{4}, \mathbf{0}, \mathbf{3}, \mathbf{5}, 8, 9, 6]$$

Procedimento QUICKSORT

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

QUICKSORT(A, p, r)

```
1 if p < r then
2 |   q ← PARTITION(A, p, r)
3 |   QUICKSORT(A, p, q − 1)
4 |   QUICKSORT(A, q + 1, r)
```

Considere a entrada:

$$A = [8, 1, 6, 4, 0, 3, 9, 5]$$

Chamada recursiva da linha 4 para a parte em vermelho:

$$A = [\mathbf{0}, \mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{8}, \mathbf{9}, \mathbf{6}]$$

Complexidade do Quicksort

Complexidade de tempo ($n = r - p + 1$)

Quicksort(A, p, r)	Tempo
1 **if** $p < r$	
2 $\quad q \leftarrow$ Partition(A, p, r)	
3 $\quad$ Quicksort$(A, p, q-1)$	
4 $\quad$ Quicksort$(A, q+1, r)$	

Complexidade do QUICKSORT

Complexidade de tempo ($n = r - p + 1$)

QUICKSORT(A, p, r)	Tempo
1 **if** $p < r$	$\Theta(1)$
2 $\quad q \leftarrow$ PARTITION(A, p, r)	$\Theta(n)$
3 $\quad$ QUICKSORT$(A, p, q-1)$	$T(k)$
4 $\quad$ QUICKSORT$(A, q+1, r)$	$T(n-k-1)$

Portanto

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

$$0 \leq k = q - p \leq n - 1.$$

Complexidade do QUICKSORT

O tempo de execução do QUICKSORT depende do particionamento dos subvetores.

Recorrência

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ \Theta(1) & n = 1 \\ T(k) + T(n-k-1) + \Theta(n) & n \geq 2 \end{cases}$$

Pior caso

Quando os subvetores estão completamente desbalanceados: um com **0** elementos e outro com **n-1**.

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

Complexidade do Quicksort

Análise no melhor caso

- Ocorre quando os subarrays estão balanceados.
- Um subarray tem tamanho $\lfloor n/2 \rfloor$ e o outro tem tamanho $\lceil n/2 \rceil - 1$.
- Obtemos a recorrência:

$$\begin{aligned} T(n) &\leq 2\,T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \qquad \text{caso 2 do teorema mestre.} \end{aligned}$$

Complexidade do Quicksort

Particionamento constante

- O tempo médio de execução do Quicksort é muito mais próximo do melhor caso do que do pior caso.
- Suponha que o algoritmo de particionamento sempre produza uma divisão na proporção 9 para 1.
- Obtemos a recorrência:

$$\begin{aligned} T(n) &\leq T(9n/10) + T(n/10) + \Theta(n) \\ &= O(n \lg n). \end{aligned}$$

- Por que? Vejamos a árvore de recursão.

Complexidade do QUICKSORT

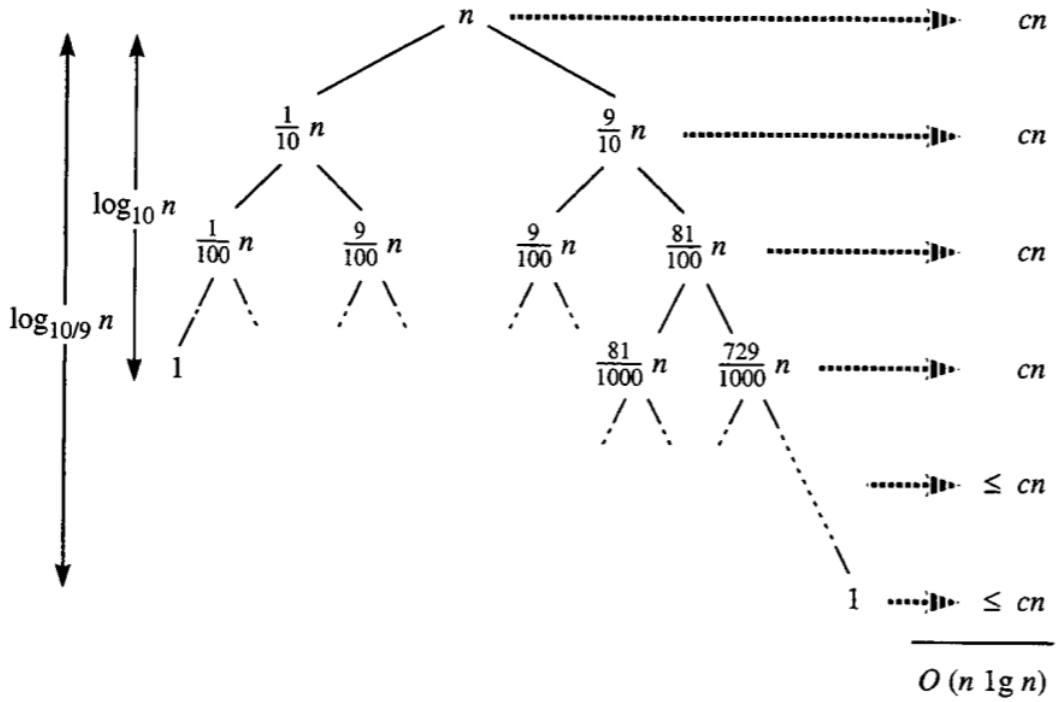


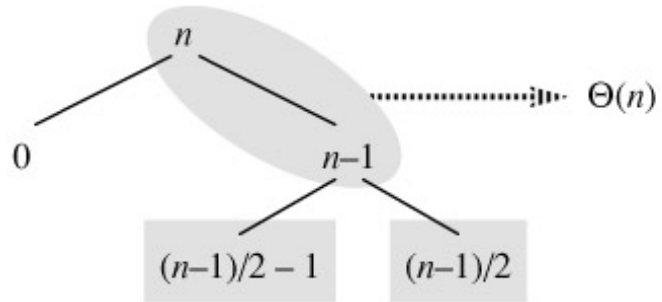
FIGURA 7.4 Uma árvore de recursão para QUICKSORT, na qual PARTITION sempre produz uma divisão de 9 para 1, resultando no tempo de execução $O(n \lg n)$. Os nós mostram tamanhos de subproblemas, com custos por nível à direita. Os custos por nível incluem a constante c implícita no termo $\Theta(n)$

Observação importante

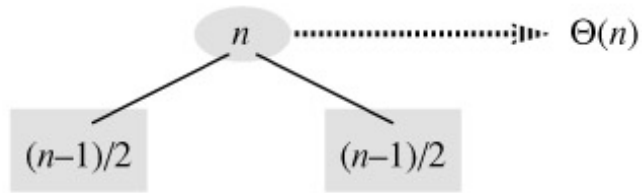
Desde que seja constante, a base do logaritmo não importa para a notação assintótica. Qualquer divisão de proporção constante gerará uma árvore de recursão de profundidade $\Theta(\lg n)$.

Intuição para o caso médio

- A proporção de divisões não será sempre constante.
- Suponha que que as divisões boas e ruins se alternem.



(a)



(b)

- A combinação de divisões boas e ruins resulta em um tempo esperado $\Theta(n \lg n)$, mas com uma constante maior escondida pela notação Θ.

Versão aleatória do QUICKSORT

- Para explorar o caso médio, assumiremos que todas as permutações de entrada são igualmente possíveis (o que nem sempre ocorre – ver Exercício 7.2-4).
- Para corrigir esta situação, adicionamos aleatoriedade ao QUICKSORT.
- A ideia é não usar sempre $A[r]$ como pivô. Ao invés, escolhemos um elemento do vetor aleatoriamente.
- Como o pivô é escolhido aleatoriamente, esperamos que a divisão do vetor de entrada seja equilibrada na média.

Versão aleatória do Quicksort

```
RANDOMIZED-PARTITION(A, p, r)
1 i ← RANDOM(p, r)
2 SWAP(A[r], A[i])
3 return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
1 if p < r then
2 |   q ← RANDOMIZED-PARTITION(A, p, r)
3 |   RANDOMIZED-QUICKSORT(A, p, q − 1)
4 |   RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Análise do tempo esperado do RANDOMIZED-QUICKSORT

- QUICKSORT e RANDOMIZED-QUICKSORT diferem apenas na maneira como o pivô é escolhido.
- Uma vez que um elemento é escolhido como pivô, ele nunca será incluído novamente em futuras chamadas de PARTITION, portanto, ocorrem ao todo no máximo n chamadas de PARTITION.
- Cada chamada de PARTITION consome tempo proporcional ao número de iterações do laço **for** (linhas 3–6).
- Limitaremos este consumo pela quantidade de vezes que a linha 4 é executada.

Relembrando algumas definições

Variável aleatória indicadora

Dado um espaço amostral S e um evento A, uma **variável aleatória indicadora** $I\{A\}$ para o evento A é definida como:

$$I\{A\} = \begin{cases} 1 & \text{se } A \text{ ocorre,} \\ 0 & \text{se } A \text{ não ocorre.} \end{cases}$$

Valor esperado (esperança)

A esperança de uma variável aleatória discreta X é dada por:

$$E[X] = \sum_x x \cdot Pr\{X = x\}$$

Linearidade da esperança

Para qualquer coleção finita de variáveis aleatórias discretas $\{X_1, \ldots, X_n\}$ com esperanças finitas

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

Análise do tempo esperado do RANDOMIZED-QUICKSORT

Objetivo

Seja X o total de comparações feitas em **todas** as chamadas do PARTITION, queremos determinar este valor de X.

Entendendo quando o algoritmo compara dois elementos?

- Renomeie os elementos de A para $z_1, z_2, \ldots z_n$, com z_i sendo o i-ésimo menor elemento.
- Seja $Z_{ij} = \{z_i, z_{i+1}, \ldots z_j\}$ os elementos entre z_i e z_j inclusive.
- Quando o algoritmo compara z_i e z_j?

Análise do tempo esperado do RANDOMIZED-QUICKSORT

Objetivo

Seja X o total de comparações feitas em **todas** as chamadas do PARTITION, queremos determinar este valor de X.

Entendendo quando o algoritmo compara dois elementos?

- Renomeie os elementos de A para $z_1, z_2, \ldots z_n$, com z_i sendo o i-ésimo menor elemento.
- Seja $Z_{ij} = \{z_i, z_{i+1}, \ldots z_j\}$ os elementos entre z_i e z_j inclusive.
- Quando o algoritmo compara z_i e z_j?
  - Quando z_i ou z_j são pivôs.
  - Cada par de elementos é comparado no máximo uma única vez.
  - Depois de usado o pivô, nunca mais o elemento é comparado.

Análise do tempo esperado do RANDOMIZED-QUICKSORT

Usaremos a seguinte variável aleatória indicadora:

$$X_{ij} = I\{z_i \text{ é comparado com } z_j\}$$

Como cada par é comparado no máximo uma vez, podemos caracterizar o número total de comparações efetuadas pelo algoritmo como:

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

Aplicando a esperança em ambos os lados temos:

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

Análise do tempo esperado do RANDOMIZED-QUICKSORT

Usando a linearidade da esperança temos:

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ ser comparado com } z_j\}.
\end{aligned}
$$

Precisamos então computar: $Pr\{z_i \text{ ser comparado com } z_j\}$

Análise do tempo esperado do RANDOMIZED-QUICKSORT

A comparação entre z_i e z_j só ocorre quando um deles é escolhido como pivô do conjunto Z_{ij}. (Assumindo que todos os elementos são distintos.)

Um exemplo

- Suponha a execução do RANDOMIZED-QUICKSORT para elementos de 1 a 10 (em qualquer ordem).
- Suponha que o primeiro pivo é o 7.
- Então o PARTITION separa os conjuntos em $\{1, 2, 3, 4, 5, 6\}$ e $\{8, 9, 10\}$.
- Nenhum membro do primeiro conjunto será comparado com um membro do segundo conjunto.

Análise do tempo esperado do RANDOMIZED-QUICKSORT

- Qualquer elemento de Z_{ij} (em uma mesma partição) pode ser escolhido como pivô.
- Z_{ij} possui $j - i + 1$ elementos.
- Como o pivô é escolhido aleatoriamente, a probabilidade de cada elemento ser escolhido como pivô é: $1/(j - i + 1)$

Portanto:

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

Análise do tempo esperado do RANDOMIZED-QUICKSORT

Continuando o desenvolvimento da equação do valor esperado, usando substituição de variáveis ($k = j - i$) e limitando pela série harmônica, temos:

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

Exercícios

7.1-1 Usando a figura 7.1 (ver livro-texto) como modelo, ilustre a operação de PARTITION sobre o array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$.

7.1-2 Que valor de q PARTITION retorna quando todos os elementos no arranjo $A[p \ldots r]$ têm o mesmo valor? Modifique PARTITION de forma que $q = (p + r)/2$ quando todos os elementos no array $A[p \ldots r]$ têm o mesmo valor.

7.1-3 Forneça um breve argumento mostrando que o tempo de execução de PARTITION sobre um subarray de tamanho n é $\Theta(n)$.

7.1-4 De que maneira você modificaria QUICKSORT para fazer a ordenação em ordem não crescente?

Exercícios

7.2-1 Use o método de substituição para provar que a recorrência $T(n) = T(n-1) + \Theta(n)$ tem a solução $T(n) = \Theta(n^2)$.

7.2-2 Qual o tempo de execução de QUICKSORT quando todos os elementos do array A têm o mesmo valor.

7.2-3 Mostre que o tempo de execução de QUICKSORT é $\Theta(n^2)$ quando o array A contém elementos distintos e está ordenado em ordem decrescente.

Exercícios

7.3-1 Por que analisamos o desempenho do caso médio de um algoritmo aleatório e não o seu desempenho no pior caso?

7.3-2 Durante a execução do procedimento RANDOMIZED-QUICKSORT, quantas chamadas são feitas ao gerador de números aleatórios RANDOM no pior caso? E no melhor caso? Dê a resposta em termos da notação Θ.

Referências

- Thomas H. Cormen et al. Introdução a Algoritmos. 2^a edição em português. Capítulo 7.
- Robert Sedgewick and Kevin Wayne. 2011. Algorithms (4th ed.). Addison-Wesley Professional.