Estruturas de Dados e Básicas I - DIM0119

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 25, ramal 239, selan@dimap.ufrn.br
UFRN

5 de abril de 2018

Algoritmos de Ordenação — Conteúdo

Observações sobre *Quick sort*

- É um algoritmo que segue o paradigma divisão-e-conquista.
- É um algoritmo recursivo.
- É um algoritmo instável.
- Complexidade temporal:
  - Melhor caso ou na média $\Rightarrow O(n \lg n)$.
  - Pior caso $\Rightarrow O(n^2)$.
    - Pode ser evitado!

Ordenação por Quick sort

Algoritmo **Quick Sort** — pseudocódigo

quicksort & quicksort_aux

Pre-condição : Arranjo com n elementos.
Pós-condição : Permutação do arranjo original tal que $A[0] \leqslant A[1] \leqslant A[2] \leqslant \cdots \leqslant A[n-1]$.

procedimento quicksort(A: **arranjo de ref inteiro**; n: **inteiro**)
  quicksort_aux(A, 0, $n-1$)

procedimento quicksort_aux(A: **arranjo de ref inteiro**; l: **inteiro**; r: **inteiro**)
  se $l < r$ **então** # Temos pelo menos 2 elementos?
    $q \leftarrow$ partição(A, l, r) # Realizar partição.
    quicksort_aux(A, l, $q-1$) # Particionar subvetor esquerdo.
    quicksort_aux(A, $q+1$, r) # Particionar subvetor direito.

Ordenação por Quick sort

Algoritmo **Quick Sort** — pseudocódigo

partição

Pre-condição : Arranjo possui 2 ou mais elementos, ou $r > l$.
Pós-condição : Seja $q \in \{l, \ldots, r\}$ o índice retornado, então $A[i] \leqslant A[q]$ $\forall i \in \{l, \ldots, q-1\}$ e $A[q] \leqslant A[j]$ $\forall j \in \{q+1, \ldots, r\}$.

função partição(A: **arranjo de ref inteiro**; l: **inteiro**; r: **inteiro**): **inteiro**
  var x: **inteiro** $\leftarrow A[r]$ # seleção pivô.
  var i: **inteiro** $\leftarrow l - 1$ # começa fora do vetor.
  var j: **inteiro** # controla laço que percorrerá vetor.
  para $j \leftarrow l$ **até** $r - 1$ **faça** # todos menos o último, que é o pivô
    se compara($A[j]$, x) $\neq 1$ **então** # Processando elemento j: $A[j] \leqslant x$?
      $i \leftarrow i + 1$ # Avançar a região dos $\leqslant$ do que pivô.
      $A[i] \leftrightarrow A[j]$ # Traz elemento processado p/ a região dos menores
  $A[i+1] \leftrightarrow A[r]$ # Traz pivô para o ‘meio’ entre regiões.
  retorna $i + 1$ # Retorna o índice do pivô.

Ordenação por Quick sort

Análise de complexidade (1)

procedimento quicksort_aux(A: **arranjo de ref inteiro**; l: **inteiro**; r: **inteiro**)
  se $l < r$ **então** # c_1
    $q \leftarrow$ partição(A, l, r) # ???
    quicksort_aux(A, r, $q - 1$)
    quicksort_aux(A, $q + 1$, l)

Ordenação por Quick sort

Análise de complexidade (2)

função partição(A: **arranjo de ref inteiro**; l: **inteiro**; r: **inteiro**): **inteiro**
  var x: **inteiro** $\leftarrow A[r]$ # c_1
  var i: **inteiro** $\leftarrow l - 1$ # c_2
  var j: **inteiro**
  para $j \leftarrow l$ **até** $r - 1$ **faça** # Laço roda $(r-1) - l + 1 \approx n$.
    se compara($A[j]$, x) $\neq 1$ **então** # operação dominante: c_3
      $i \leftarrow i + 1$ # c_4
      $A[i] \leftrightarrow A[j]$ # c_5
  $A[i+1] \leftrightarrow A[r]$ # c_6
  retorna $i + 1$ # c_7

- $T(n) = (c_3 + c_4 + c_5) \cdot n + (c_1 + c_2 + c_6 + c_7)$.
- $T(n) = a \cdot n + b$.

Ordenação por Quick sort

Melhor caso

procedimento quicksort_aux(A: **arranjo de ref inteiro**; l: **inteiro**; r: **inteiro**)
  se $l < r$ **então** # c_1
    $q \leftarrow$ partição(A, l, r) # an (chamada) $+c_2$ (atribuição)
    quicksort_aux(A, r, $q - 1$) # $T(n/2)$
    quicksort_aux(A, $q + 1$, l) # $T(n/2)$

- No melhor caso a partição retorna subvetores de tamanho $\frac{n}{2}$ ou $\frac{n}{2} - 1$.

$$[6,\ 8,\ 3,\ 2,\ 1,\ 7,\ 5,\ \mathbf{4}] \Rightarrow [1,\ 2,\ 3,\ \mathbf{4},\ 8,\ 5,\ 7,\ 6].$$

- $T(n) = 2T(n/2) + an + c_2 + c_1$.
- Todas as partições serão balanceadas.
- $T(n) = 2T(n/2) + cn$ (desprezando e renomeando constantes):

$$T(n) = \begin{cases} 2T(n/2) + cn & \text{se } n > 1, \\ c_1 & n = 1. \end{cases}$$

Ordenação por Quick sort
Melhor caso

- ▻ Resolvendo a relação de recorrência

$$
\begin{aligned}
T(1) &= c_1 \\
T(n) &= 2T(\tfrac{n}{2}) + cn. \quad \text{passo 1} \\
&= 2\{2T(\tfrac{n}{4}) + c\tfrac{n}{2}\} + cn \\
&= 4T(\tfrac{n}{4}) + 2cn \quad \text{passo 2} \\
&= 4\{2T(\tfrac{n}{8}) + c\tfrac{n}{4}\} + 2cn \\
&= 8T(\tfrac{n}{8}) + 3cn \quad \text{passo 3} \\
&\ldots \\
&= 2^k T(\tfrac{n}{2^k}) + kcn \quad k\text{-ésimo passo}
\end{aligned}
$$

Ordenação por Quick sort
Melhor caso

- ▻ Resolvendo a relação de recorrência

$$
\begin{aligned}
T(1) &= c_1 \\
T(n) &= 2T(\tfrac{n}{2}) + cn. \quad \text{passo 1} \\
&= 4T(\tfrac{n}{4}) + 2cn \quad \text{passo 2} \\
&= 8T(\tfrac{n}{8}) + 3cn \quad \text{passo 3} \\
&= 2^k T(\tfrac{n}{2^k}) + kcn \quad k\text{-ésimo passo}
\end{aligned}
$$

- ▻ Eventualmente a redução expressará $T(n)$ em função de $T(1)$, logo

$$\frac{n}{2^k} = 1 \;\Rightarrow\; 2^k = n \;\Rightarrow k = \log_2 n$$

- ▻ $T(n) = 2^{\log_2 n} T(1) + cn(\log_2 n) = nc_1 + cn \log n$
- ▻ $T(n) = \Omega(n \log n)$!

Ordenação por Quick sort
Pior caso

```
procedimento quicksort_aux(A: arranjo de ref inteiro; l: inteiro; r: inteiro)
    se l < r então # c1
        q ← partição(A, l, r)# an (chamada) +c2 (atribuição)
        quicksort_aux(A, r, q − 1)                          # T(n − 1)
        quicksort_aux(A, q + 1, l)                          # c3
```

- ▻ No pior caso uma partição será sempre vazia.
- ▻ $T(n) = T(n-1) + an + c_2 + c_1 + c_3$.
- ▻ $T(n) = T(n-1) + cn$ (desprezando e renomeando constantes):

$$
\begin{cases}
T(n) = T(n-1) + cn \\
T(1) = c_1.
\end{cases}
$$

Ordenação por Quick sort
Relação de recorrência

$$
\begin{aligned}
T(1) &= c_1 \\
T(n) &= T(n-1) + cn. \quad \text{passo 1} \\
&= \{T(n-2) + c(n-1)\} + cn \\
&= T(n-2) + 2cn - c \quad \text{passo 2} \\
&= \{T(n-3 + c(n-2)\} + 2cn - c \\
&= T(n-3) + 3cn - 3c \quad \text{passo 3} \\
&= \{T(n-4) + c(n-3)\} + 3cn - 3c \\
&= T(n-4) + 4cn - 6c \quad \text{passo 4} \\
&= \{T(n-5) + c(n-4)\} + 4cn - 6c \\
&= T(n-5) + 5cn - 10c \quad \text{passo 5} \\
&\ldots \\
&= T(n-k) + kcn - (1 + 2 + \cdots + k - 1)c \quad k\text{-ésimo passo} \\
&= T(n-k) + kcn - \tfrac{k(k-1)}{2}c \quad k\text{-ésimo passo}
\end{aligned}
$$

$$
\begin{aligned}
T(1) &= c_1\\
T(n) &= T(n-1) + cn. \quad \text{passo 1}\\
&= T(n-k) + kcn - \tfrac{k(k-1)}{2}c
\end{aligned}
$$

- Considerando que vamos chegar em $T(1)$, temos:

$$
\begin{aligned}
n-k &= 1 \ \Rightarrow \ n = k.\\
T(n) &= T(1) + cn^2 - \tfrac{n(n-1)}{2}c\\
&= c_1 + cn(n - \tfrac{(n-1)}{2})\\
&= c_1 + cn(\tfrac{n+1}{2})\\
&= c\tfrac{n^2}{2} + c\tfrac{n}{2} + c_1.
\end{aligned}
$$

- Portanto, $T(n) = O(n^2)$.
- Pode ser resolvido se escolhermos o pivô aleatoriamente.

Referências

- Paulo A. Azeredo
  Métodos de Classificação de Dados,
  Editora Campus, 1996.
- Robert L. Kruse, Alexander J. Ryba
  Data Structures and Program Design in C++, Third Edition, **Cap. 8**.
  Prentice Hall, New Jersey/USA, Addison Wesley, 2000.