QuickSort

Pedro Henrique Del Bianco Hokama
28 de Agosto de 2019

Referências:

- Notas de aulas fortemente baseadas no curso: Stanford Algorithms by Tim Roughgarden
  - `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  - Vídeos: 5.1 até 5.4
- CLRS: Cap 7
- Cormen T., Desmistificando Algoritmos Cap 3

1 Introdução

Porque ver e rever o QuickSort:

- Algoritmo famoso.
- Muito usado da prática.
- Análise interessante.
- Memória extra constante

Definição 1.1: Problema de Ordenação:

Entrada: Um arranjo A contendo n inteiros em uma ordem arbitrária.

Saída: Um arranjo B com os mesmos elementos, ordenados de forma crescente.

Exercício 1.1: Quais alterações necessárias se o a entrada tiver elementos repetidos.

1.1 Partição

O QuickSort depende fortemente da operação de Partição cuja a ideia é particionar o arranjo em torno de um pivô:

- Escolha um pivô.
- Reorganize os elementos do arranjo de forma que:
  - A esquerda do pivô tenha os elementos menores que o pivô.
  - A direita do pivô tenha os elementos maiores que o pivô.

Exemplo 1.1: (3, 8, 2, 5, 1, 4, 7, 6)

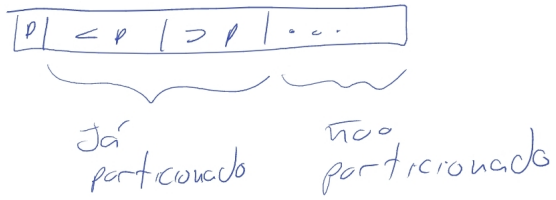
Uma partição seria: (2, 1, 3, 6, 7, 4, 5, 8)

Observe que depois de uma partição o pivô estará na sua posição correta. Note também que não nos importamos com a posição relativa de cada uma das partes esquerda e direita.

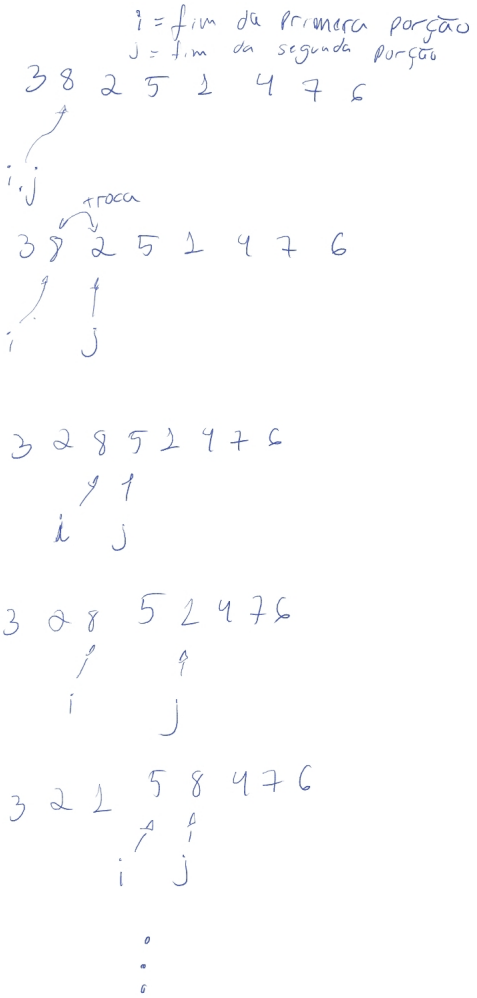
Exercício 1.2: Como particionar em $O(n)$ usando memória extra?

Podemos fazer a Partição em tempo $O(n)$ sem usar memória extra.
Ideia:

- Manter uma porção do arranjo já particionado, e o restante ainda não particionado



- Percorrer uma única vez o arranjo



Algoritmo 1: Partição

Entrada: Um arranjo A, índices l e r
Saída: O mesmo arranjo mas particionado
início
  $p = A[l]$;
  $i = l + 1$;
  para $j = l + 1$ *até* r **faça**
    se $A[j] < p$ **então**
      Troca $A[j]$ e $A[i]$;
      $i = i + 1$;
    fim
  fim
  Troca $A[l]$ e $A[i - 1]$;
  devolva A;
fim

1.2 Corretude do algoritmo partição

Invariante de laço: Os elementos $A[l+1], \ldots, A[i-1]$ são todos menores que o pivô, e os elementos $A[i], \ldots, A[j-1]$ são todos maiores que o pivô.

- **Inicialização:** Por vacuidade, antes da primeira iteração a invariante vale.
- **Manutenção:** Em cada iteração verificamos se $A[j]$ é menor que p e nesse caso passamos ele para a primeira porção, incrementando o i. E dessa forma a invariante de mantêm.
- **Termino:** Ao final do laço trocamos o pivô pelo ultimo elemento da primeira porção (ou ele mesmo caso a primeira porção seja vazia) e o arranjo estará particionado.

1.3 QuickSort

Algoritmo 2: QuickSort

Entrada: Um arranjo A, o comprimento do arranjo n
Saída: Um arranjo com os mesmos elementos de A porém ordenados
início
  se $n \leq 1$ **então** devolva A;
  p = EscolhePivo(A, n);
  Particionar A em torno de p;
  Recursivamente ordenar a parte esquerda;
  Recursivamente ordenar a parte direita;
  devolva A;
fim

Exercício 1.3: Qual o tempo de execução da fase de combinação?

1.4 Corretude do algoritmo QuickSort

Podemos provar o QuickSort por indução no número de elementos. Primeiramente vamos revisar o que é uma prova por indução, que utiliza o seguinte axioma:

Definição 1.2: Seja $P(n)$ uma sentença aberta sobre $\mathbb{N}$. Suponha que:

1. $P(0)$ é verdade, e
2. Sempre que $P(k)$ é verdade, para algum $k \in \mathbb{N}$, temos que $P(k+1)$ é verdade.

Então $P(n)$ é verdade para todo $n \in \mathbb{N}$.

Para usar esse método podemos utilizar o seguinte roteiro:

- *Base da Indução*: Provar que $P(0)$ é verdade.
- *Hipótese de Indução*: Supor que para algum $k \in \mathbb{N}$, $P(k)$ é verdade.
- *Passo da Indução*: Provar que $P(k+1)$ é verdade.

Teorema 1.1: O algoritmo QuickSort está correto.

Prova: Iremos provar a corretude do QuickSort por indução no comprimento n do arranjo. Considere a seguinte sentença aberta $P(n)$: "O QuickSort corretamente ordena arranjos de comprimento n.

- *Base da Indução*: Quando $n = 1$ o algoritmo não faz nada e o vetor está trivialmente ordenado. Portando $P(1)$ é verdade.
- *Hipótese de Indução*: $P(k)$ é verdade para $k < n$
- *Passo da Indução*: Agora queremos provar $P(k+1)$ é verdade.
  - O algoritmo escolhe algum pivô p e particiona o arranjo
  - O pivô termina já em seu lugar correto
  - O algoritmo recursivamente ordena a primeira e a segunda porção que necessariamente são menores que k, logo pela hipótese de indução, são ordenadas corretamente.

■