Análise de Algoritmos na Prática: O Algoritmo *InsertionSort*

Paulo Henrique Ribeiro Gabriel

Faculdade de Computação
Universidade Federal de Uberlândia

phrg@ufu.br

Nesta aula

- Ordenação
- Ordenação por inserção
- Análise

Ordenação

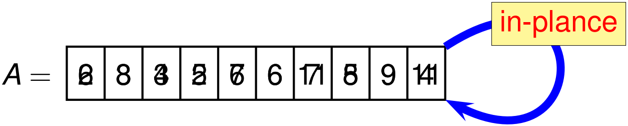
- **Entrada:** uma sequência $A = \langle a_1, a_2, \ldots, a_n \rangle$

  Saída: uma sequência $\langle b_1, b_2, \ldots, b_n \rangle$ tal que

  - $\langle b_1, b_2, \ldots, b_n \rangle$ é uma *permutação* de $\langle a_1, a_2, \ldots, a_n \rangle$
  - $\langle b_1, b_2, \ldots, b_n \rangle$ está *ordenada*

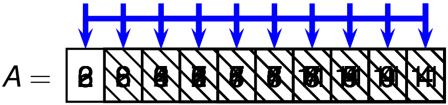
$$b_1 \leq b_2 \leq \cdots \leq b_n$$

- Tipicamente, A é implementado como um *array*



Ordenação por Inserção

- **Ideia:** similar a ordenação de cartas na mão
  - Percorra a sequência da esquerda para a direita
  - Pegue o valor da posição atual a_j
  - Insira esse valor na sua posição correta na dentro da sequência $\langle a_1, a_2, \ldots a_{j-1}\rangle$ de modo a manter a subsequência ordenada $\langle a_1, a_2, \ldots a_j\rangle$



Ordenação por Inserção (2)

```
INSERTIONSORT(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- O INSERTIONSORT é *correto*?
- Qual a complexidade do INSERTIONSORT?
- Podemos fazer melhor?

Complexidade do InsertionSort

```
InsertionSort(A)
1  for i = 2 to comprimento(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          troque a chave A[j] com A[j − 1]
5          j = j − 1
```

- Laço externo (linhas 1–5) é executado exatamente $n - 1$ vezes (lembrando que $n = \mathit{comprimento}(A)$)
- E o laço interno (linhas 3–5)?
  - Melhor caso, pior caso e caso médio?

Complexidade do InsertionSort (2)

```
InsertionSort(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- **Melhor caso:** o laço interno *nunca* é executado
  - Quando isso ocorre?

- **Pior caso:** o laço interno é executado exatamente $j - 1$ vezes para *toda* iteração do laço externo
  - Quando isso ocorre?

Complexidade do InsertionSort (3)

- A complexidade de pior caso ocorre quando o laço interno é executado exatamente $j-1$ vezes, portanto

$$T(n) = \sum_{j=2}^{n} (j-1)$$

$T(n)$ é a *série aritmética* $\sum_{k=1}^{n-1} k$, portanto

$$T(n) = \frac{n(n-1)}{2}$$

$$\boxed{T(n) = O(n^2)}$$

- O melhor caso é $T(n) = \Omega(n)$
- Caso médio é $T(n) = \Theta(n^2)$ (por quê?)

Corretude

- O INSERTIONSORT sempre termina para valores de entrada válidos?
- Em caso afirmativo, ele satisfaz as condições de um problema de ordenação (ou seja, ele resolve o problema)?
  - A contém uma *permutação* de valores iniciais de A
  - A está *ordenado*, ou seja $A[1] \leq A[2] \leq \cdots \leq A[comprimento(A)]$
- Nós queremos uma prova formal de corretude

A lógica dos passos algorítmicos

Exemplo:

```
SORTTWO(A)
// A deve ser um array de 2 elementos
if A[1] > A[2]
    t = A[1]
    A[1] = A[2]
    A[2] = t
```

Invariantes de Laço

- Formulamos uma condição de *invariante de laço C*
  - *C* deve permanecer verdadeiro *através* de um laço
  - *C* é relevante para a definição do problema: devemos usar *C* ao final do laço para provar a corretude do resultado
- Assim, precisamos apenas provas que o algoritmo termina

Invariantes de Laço (2)

- Formulação:
  - *A invariante deve refletir a estrutura do algoritmo*
  - Deve ser a base para provar que a solução está correta
- Prova de validade (ou seja, C é, de fato, uma invariante de laço?): tipicamente é uma *prova por indução*
  - Inicialização: devemos provar que
    A invariante C é verdadeira antes de entrarmos no laço
  - Manutenção: devemos provar que
    Se *C é verdadeira no início de um ciclo* ***então*** *ela permanece verdadeira após um ciclo*

Invariante de Laço para o INSERTIONSORT

```
INSERTIONSORT(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- A ideia central é inserir $A[i]$ em $A[1\,..\,i-1]$ de modo a manter uma *subsequência ordenada* $A[1\,..\,i]$
- Invariante: (laço externo) o subarray $A[1\,..\,i-1]$ é composto por elemento originalmente em $A[1\,..\,i-1]$ *em ordem*

Invariante de Laço para o InsertionSort (2)

```
InsertionSort(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- **Inicialização:** $j = 2$, então $A[1\,..\,j-1]$ é o único elemento $A[1]$
  - $A[1]$ contém o elemento original em $A[1]$
  - $A[1]$ está ordenado de maneira trivial

Invariante de Laço para o INSERTIONSORT (3)

```
INSERTIONSORT(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- **Manutenção:** informalmente, se $A[1\,..\,i-1]$ é uma permutação do array original $A[1\,..\,i-1]$ e $A[1\,..\,i-1]$ está ordenado (invariante), então *se* entrarmos no laço interno:
  - Alteramos o subarray $A[k\,..\,i-1]$ em uma posição para a direita
  - Insira a *chave*, que está originalmente em $A[i]$ em sua posição correta $1 \leq k \leq i-1$, de maneira ordenada

Invariante de Laço para o InsertionSort (4)

```
InsertionSort(A)
for i = 2 to comprimento(A)
    j = i
    while j > 1 and A[j − 1] > A[j]
        troque a chave A[j] com A[j − 1]
        j = j − 1
```

- **Término:** o InsertionSort termina com $i = comprimento(A) + 1$; a invariante define que
  - $A[1 \mathinner{..} i-1]$ é uma permutação do array original $A[1 \ldots i-1]$
  - $A[1 \mathinner{..} i-1]$ está ordenado

  Dada a condição de parada, $A[1 \mathinner{..} i-1]$ é todo array A

  Portanto, o InsertionSort é *correto!*

Sumário

- Seja um problema P e um algoritmo A
  - P formalmente define uma condução de *corretude*
  - Suponha, por simplicidade, que A é composto por um laço

---

1. Formule uma invariante C
2. **Inicialização** (para toda entrada válida)
   - Prove que C está correto antes da primeira execução da primeira instrução do laço
3. **Manutenção** (para toda entrada válida)
   - Prove que, se C permanece correto para a primeira instrução do laço, então C estará correto ao final do laço
4. **Encerramento** (para toda entrada válida)
   - Prove que o laço termina, com alguma condição de saída X
5. Prove que $X \wedge C \Rightarrow P$, o que significa que A é correto

Exercício: Análise do SelectionSort

```
SelectionSort(A)
n = comprimento(A)
for i = 1 to n − 1
    menor = i
    for j = i + 1 to n
        if A[j] < A[menor]
            menor = j
    troque a chave A[i] com A[menor]
```

- Corretude?
  - Invariante de laço?
- Complexidade?
  - Melhor caso, pior caso e caso médio?

Exercício: Análise do Bubblesort

```
BUBBLESORT(A)
for i = 1 to comprimento(A)
    for j = comprimento(A) downto i + 1
        if A[j] < A[j − 1]
            troque a chave A[j] com A[j − 1]
```

- Corretude?
  - Invariante de laço?
- Complexidade?
  - Melhor caso, pior caso e caso médio?