Verão 2019 - TÓPICOS DE PROGRAMAÇÃO

Prof. Dr. Leônidas O. Brandão (coordenador)

Profa. Dra. Patrícia Alves Pereira (ministrante)

Prof. Bernardo (Monitor)

1

Problema de ordenação

Problema:

Rearranjar um vetor $A[1 \dots n]$ de inteiros de modo que fique em ordem crescente.

Ou simplesmente:

Problema:

Ordenar um vetor $A[1 \dots n]$ de inteiros.

Insertion sort
Ordenação por inserção

Ordenação por inserção

- **Idéia básica:** a cada passo mantemos o subvetor $A[1 \ldots j-1]$ ordenado e inserimos o elemento $A[j]$ neste subvetor.
- Repetimos o processo para $j = 2, \ldots, n$ e ordenamos o vetor.

Ordenação por inserção - exemplo

5 - 3 - 1 - 4 - 2

Inicialmente considera-se o primeiro elemento do arranjo como se ele estivesse ordenado;

ele será considerado o subarranjo ordenado inicial.

Ordenação por inserção - exemplo

Agora o elemento imediatamente superior ao o subarranjo ordenado, no o exemplo o número 3, deve se copiado para uma variável auxiliar qualquer.

5 - 3 - 1 - 4 - 2

Após copiá-lo, devemos percorrer o subarranjo a partir do último elemento para o primeiro.
Ao encontrar, devemos ser copiar o elemento para a posição imediatamente superior ao o subarranjo ordenado.

3 - 5 - 1 - 4 - 2

Ordenação por inserção - exemplo

3 - 5 - 1 - 4 - 2

Verifique que o subarranjo ordenado possui agora dois elementos.
Vamos repetir o processo anterior para que se continue a ordenação.

1 - 3 - 5 - 4 - 2

1 - 3 – 4 - 5 - 2

1 - 2 - 3 – 4 - 5

Ordenação por inserção - pseudo código

```
ORDENA-POR-INSERÇÃO(A, n)
  para j ← 2 até n faça
      chave ← A[j]
      ▷ Insere A[j] no subvetor ordenado A[1..j − 1]
      i ← j − 1
      enquanto i ≥ 1  e A[i] > chave faça
          A[i + 1] ← A[i]
          i ← i − 1
      A[i + 1] ← chave
```

Determine a Finitude e Corretude ?

Qual é a situação do melhor e pior caso?

Determine as complexidades para os dois casos.

Ordenação por inserção - Finitude

```
ORDENA-POR-INSERÇÃO(A, n)
1    para j ← 2 até n faça
     . . .
4        i ← j − 1
5        enquanto i ≥ 1 e A[i] > chave faça
6            . . .
7            i ← i − 1
8        . . .
```

No laço **enquanto** na linha 5 o valor de i diminui a cada iteração e o valor inicial é $i = j - 1 \geq 1$. Logo, a sua execução pára em algum momento por causa do teste condicional $i \geq 1$.

O laço na linha 1 evidentemente pára (o contador j atingirá o valor $n + 1$ após $n - 1$ iterações).

Portanto, o algoritmo **pára**.

Relembrando:
Invariantes de laço e provas de corretude

- Definição: um invariante de um laço é uma propriedade que relaciona as variáveis do algoritmo a cada execução completa do laço.
- Ele deve ser escolhido de modo que, ao término do laço, tenha-se uma propriedade útil para mostrar a corretude do algoritmo.
- A prova de corretude de um algoritmo requer que sejam encontrados e provados invariantes dos vários laços que o compõem.
- Em geral, é mais difícil descobrir um invariante apropriado do que mostrar sua validade se ele for dado de bandeja. . .

Ordenação por inserção - Corretude

ORDENA-POR-INSERÇÃO(A, n)
1 **para** $j \leftarrow 2$ **até** n **faça**
2 $\quad$ *chave* $\leftarrow A[j]$
3 $\quad$ ▷ Insere $A[j]$ no subvetor ordenado $A[1..j-1]$
4 $\quad$ $i \leftarrow j-1$
5 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] >$ *chave* **faça**
6 $\qquad$ $A[i+1] \leftarrow A[i]$
7 $\qquad$ $i \leftarrow i-1$
8 $\quad$ $A[i+1] \leftarrow$ *chave*

Invariante principal de ORDENA-POR-INSERÇÃO: (i1)

No começo de cada iteração do laço **para** das linha 1–8, o subvetor $A[1 \ldots j-1]$ está ordenado.

Ordenação inserção - Corretude por invariantes

A estratégia "típica" para mostrar a corretude de um algoritmo iterativo através de invariantes segue os seguintes passos:

1. Mostre que o invariante vale no início da primeira iteração (trivial, em geral)
2. Suponha que o invariante vale no início de uma iteração qualquer e prove que ele vale no ínicio da próxima iteração
3. Conclua que se o algoritmo pára e o invariante vale no ínicio da última iteração, então o algoritmo é correto.

Note que (1) e (2) implicam que o invariante vale no início de qualquer iteração do algoritmo. Isto é similar ao método de indução matemática ou indução finita!

Ordenação inserção - Corretude por invariantes

Vamos verificar a corretude do algoritmo de ordenação por inserção usando a técnica de **prova por invariantes de laços**.

Invariante principal: (i1)

No começo de cada iteração do laço **para** das linhas 1–8, o subvetor $A[1 \ldots j-1]$ está ordenado.

1						j				n
20	25	35	40	44	55	38	99	10	65	50

- Suponha que o invariante vale.
- Então a corretude do algoritmo é “evidente”. **Por quê?**
- No ínicio da última iteração temos $j = n+1$. Assim, do invariante segue que o (sub)vetor $A[1 \ldots n]$ está ordenado!

Ordenação inserção - Corretude por invariantes

Um invariante mais preciso: (i1')

No começo de cada iteração do laço **para** das linhas 1–8, o subvetor $A[1 \ldots j-1]$ é uma permutação ordenada do subvetor original $A[1 \ldots j-1]$.

Ordenação inserção - Corretude por invariantes

1. Validade na primeira iteração: neste caso, temos $j = 2$ e o invariante simplesmente afirma que $A[1 \ldots 1]$ está ordenado, o que é evidente.
2. Validade de uma iteração para a seguinte: segue da discussão anterior. O algoritmo empurra os elementos maiores que a chave para seus lugares corretos e ela é colocada no espaço vazio.

   Uma demonstração mais formal deste fato exige invariantes auxiliares para o laço interno **enquanto**.
3. Corretude do algoritmo: na última iteração, temos $j = n + 1$ e logo $A[1 \ldots n]$ está ordenado com os elementos originais do vetor. Portanto, o algoritmo é correto.

Ordenação inserção - Invariantes auxiliares

No início da linha 5 valem os seguintes invariantes:

(i2) $A[1 \dots i]$ e $A[i+2 \dots j]$ contém os elementos de $A[1 \dots j]$ antes de entrar no laço que começa na linha 5.

(i3) $A[1 \dots i]$ e $A[i+2 \dots j]$ são crescentes.

(i4) $A[1 \dots i] \leq A[i+2 \dots j]$

(i5) $A[i+2 \dots j] > chave$.

$$\left.\begin{array}{l}\text{Invariantes (i2) a (i5)}\\ +\text{condição de parada na linha 5}\\ +\text{atribuição da linha 7}\end{array}\right\} \Longrightarrow \text{invariante (i1')}$$

Demonstração? Mesma que antes.

Ordenação inserção - invariantes auxiliares

No início da linha 5 valem os seguintes invariantes:

(i2) $A[1 \ldots i]$ e $A[i+2 \ldots j]$ contém os elementos de $A[1 \ldots j]$ antes de entrar no laço que começa na linha 5.

(i3) $A[1 \ldots i]$ e $A[i+2 \ldots j]$ são crescentes.

(i4) $A[1 \ldots i] \leq A[i+2 \ldots j]$

(i5) $A[i+2 \ldots j] > chave$.

$$\left.\begin{array}{l}\text{Invariantes (i2) a (i5)}\\ +\text{condição de parada na linha 5}\\ +\text{atribuição da linha 7}\end{array}\right\} \Longrightarrow \text{invariante (i1')}$$

Demonstração? Mesma que antes.

Ordenação inserção - complexidade pior caso

INSERTION-SORT(A, n)	Tempo
1 **para** $j \leftarrow 2$ **até** n **faça**	$\Theta(n)$
2 $\quad chave \leftarrow A[j]$	$\Theta(n)$
3 $\quad \triangleright$ Insere $A[j]$ em $A[1 \ldots j-1]$	
4 $\quad i \leftarrow j - 1$	$\Theta(n)$
5 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**	$nO(n) = O(n^2)$
6 $\quad\quad A[i+1] \leftarrow A[i]$	$nO(n) = O(n^2)$
7 $\quad\quad i \leftarrow i - 1$	$nO(n) = O(n^2)$
8 $\quad A[i+1] \leftarrow chave$	$O(n)$

Consumo de tempo: $O(n^2)$

Ordenação inserção complexidade melhor versus pior caso

- Complexidade de tempo no pior caso: $\Theta(n^2)$
  Vetor em ordem decrescente
  $\Theta(n^2)$ comparações
  $\Theta(n^2)$ movimentações
- Complexidade de tempo no melhor caso: $\Theta(n)$
  (vetor em ordem crescente)
  $O(n)$ comparações
  zero movimentações

Selection sort
Ordenação por seleção

Ordenação por seleção

Consiste em encontrar a menor chave por pesquisa sequencial.

Encontrando a menor chave, essa é permutada com a que ocupa a posição inicial do vetor, que fica então reduzido a um elemento.

O processo é repetido para o restante do vetor, sucessivamente, até que todas as chaves tenham sido selecionadas e colocadas em suas posições definitivas.

Ordenação por seleção

5 - 3 - 1 - 4 - 2
Verifica quem é o menor e troca com a primeira posição.

1 - 3 - 5 - 4 – 2

Repete a ideia só que a partir da segunda posição:
1 - 2 - 5 - 4 – 3

e assim sucessivamente até que não exista mais menores após determinada posição:
1 - 2 - 3 - 4 - 5

Ordenação por seleção - algoritmo

```
Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.
para i = 0 até n - 2 faça
    min = i
    para j = i + 1 até n - 1 faça
        se A[j] < A[min] então
            min = j
    t = A[min]
    A[min] = A[i]
    A[i] = t
```

Sua tarefa:

Determine a Finitude e Corretude?
Qual é a situação do melhor
e pior caso?
Determine as complexidades
para os dois casos.
Implemente os dois algoritmos:
Insertion e Selection

Ordenação por seleção – complexidade

- **Complexidade Algoritmo Casos:**
- Melhor: $O(n^2)$
- Pior: $O(n^2)$