Algoritmos de Ordenação

Aula 07

Prof. Marco Aurélio Stefanes

marco@facom.ufms.br

Aula de hoje

- Introdução

Aula de hoje

- Introdução
- Ordenação por Trocas Sucessivas

Aula de hoje

- Introdução
- Ordenação por Trocas Sucessivas
- Ordenação por Inserção

Aula de hoje

- Introdução
- Ordenação por Trocas Sucessivas
- Ordenação por Inserção
- Ordenação por Seleção

Aula de hoje

- Introdução
- Ordenação por Trocas Sucessivas
- Ordenação por Inserção
- Ordenação por Seleção
- Exercícios

Algoritmos de Ordenação

- Problema tradiconal
- Dada uma seqüência de números, colocá-la em uma certa ordem (numérica ou lexicográfica)
- Na prática, os números não são valores isolados. Cada um faz parte de um **registro**.
- Cada registro contém uma chave, que é o valor a ser ordenado, e os dados satélites.
- Permutando os números, também permutamos os dados satélites.
- Supor que a entrada do problema consiste somente de números.

Definição

Dada uma seqüência de n números $\langle a_1, a_2, \ldots, a_n \rangle$, encontrar uma permutação $\langle a'_1, a'_2, \ldots, a'_n \rangle$ dessa seqüência tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

- A seguir assumimos que já existe um tipo vet_num que define um vetor de inteiros cujo limite inferior é 1 e limite superior é 100.
- Em outras palavras, omitiremos uma linha do tipo

definatipo vetor$[1..100]$ de inteiro vet_num

Bubble-sort

- Passar sequencialmente várias vezes pelos elementos $a_1, a_2, \ldots, a_n$.
- Em cada vez, a cada passo o elemento a_j é comparado com o elemento a_{j+1}
- Caso $a_j > a_{j+1}$, então eles são trocados.
- No final, a sequencia estará ordenada

Bubble-sort

Algoritmo $bubble\_sort$
inteiro i, j, n, $temp$
$vet\_num\ a$
leia n
for i de 1 até n **do**
  leia $a[i]$
for i de 1 até $n-1$ **do**
  for j de 1 até $n-1$ **do**
    if $a[j] > a[j+1]$ **then**
      $temp \leftarrow a[j]$
      $a[j] \leftarrow a[j+1]$
      $a[j+1] \leftarrow temp$
Fimalgoritmo

Bubble-sort Melhorado

Algoritmo $bubble\_sort$
inteiro i, j, n, $temp$
$vet\_num\ a$
leia n
for i de 1 até n **do**
  leia $a[i]$
for i de 1 até $n-1$ **do**
  for j de 1 até $n-i$ **do**
    if $a[j] > a[j+1]$ **then**
      $temp \leftarrow a[j]$
      $a[j] \leftarrow a[j+1]$
      $a[j+1] \leftarrow temp$
Fimalgoritmo

Insertion-sort

- Imagine a situação
- Cocê tem um conjunto de cartas em sua mão esquerda
- Com a mão direita tenta ordenar as cartas
- Por cada carta, da esquerda para direita, em sua posição correta
- Para isto, fazer inserções
- A cada passo, um elemento é inserido em sua posição correta

Insertion-sort

Algoritmo $insertion\_sort$
inteiro i, j, c, n
$vet\_num\ a$
leia n
for i de 1 até n **do**
  leia $a[i]$
for i de 2 até n **do**
  $c \leftarrow a[i]$
  $j \leftarrow i - 1$
  while $j > 0$ E $a[j] > c$ **do**
    $a[j+1] \leftarrow a[j]$
    $j \leftarrow j - 1$
  $a[j+1] \leftarrow c$
Fimalgoritmo

Selection-sort

- Princípio de funcionamento
- Tome o maior elemento e troque-o com o último elemento.
- Repita essa operação para os $n - 1$ elementos restantes.
- i.e. encontre o maior elemento entre eles e coloque-o na penúltima posição.
- Repita a operação para os $n - 2$ elementos e assim sucessivamente.

Selection-sort

```
Algoritmo selection_sort
inteiro imax, max, i, j
vet_num a
leia n
for i de 1 até n do
  leia a[i]
for i de n até 2 passo −1 do
  max ← a[1]
  imax ← 1
  for j ← 2 até i do
    if a[j] > max then
      max ← a[j]
      imax ← j
  a[imax] ← a[i]
  a[i] ← max
Fimalgoritmo
```