



Algoritmos e Programação de Computadores

Ordenação

Agenda

— — —

- O Problema da Ordenação
- Selection Sort
- Bubble Sort
- Insertion Sort
- Exercício

Ordenação

- Vamos estudar alguns algoritmos para o seguinte problema:

Dado uma coleção de elementos com **uma relação de ordem entre si**, devemos gerar uma saída com os elementos ordenados.

Ordenação

- O problema de ordenação é um dos mais básicos em computação.
 - Provavelmente é um dos problemas com o maior número de aplicações diretas ou indiretas.
- Exemplos de aplicações diretas
 - Criação de rankings; definir preferências em atendimentos por prioridade; criação de listas.
- Exemplos de aplicações indiretas
 - Otimizar sistemas de busca; manutenção de estruturas de bancos de dados.

Selection Sort

(Ordenação por Seleção)

Selection Sort (Ordenação por Seleção)

- Seja `lista` uma lista contendo números.
 - `lista = [5, 3, 2, 1, 90, 6]`.
- Devemos deixar `lista` em ordem crescente.

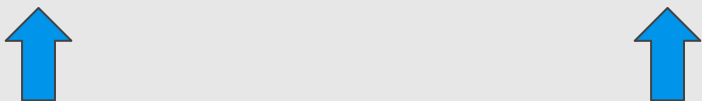
Selection Sort (Ordenação por Seleção)

- A ideia do algoritmo é a seguinte:
 - Ache o menor elemento a partir da posição 0. Troque então este elemento com o elemento da posição 0.
 - Ache o menor elemento a partir da posição 1. Troque então este elemento com o elemento da posição 1.
 - Ache o menor elemento a partir da posição 2. Troque então este elemento com o elemento da posição 2.
 - E assim sucessivamente...

Selection Sort (Ordenação por Seleção)

- Exemplo: $[5, 3, 2, 1, 90, 6]$.

Selection Sort (Ordenação por Seleção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. Acha menor: [5, 3, 2, 1, 90, 6] Faz troca: [1, 3, 2, 5, 90, 6]
- 

Selection Sort (Ordenação por Seleção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. Acha menor: [5, 3, 2, 1, 90, 6] Faz troca: [1, 3, 2, 5, 90, 6]
 - Iteração 2. Acha menor: [1, 3, 2, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]



Selection Sort (Ordenação por Seleção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. Acha menor: [5, 3, 2, 1, 90, 6] Faz troca: [1, 3, 2, 5, 90, 6]
 - Iteração 2. Acha menor: [1, 3, 2, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 3. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]



Selection Sort (Ordenação por Seleção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. Acha menor: [5, 3, 2, 1, 90, 6] Faz troca: [1, 3, 2, 5, 90, 6]
 - Iteração 2. Acha menor: [1, 3, 2, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 3. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 4. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]



Selection Sort (Ordenação por Seleção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. Acha menor: [5, 3, 2, 1, 90, 6] Faz troca: [1, 3, 2, 5, 90, 6]
 - Iteração 2. Acha menor: [1, 3, 2, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 3. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 4. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 90, 6]
 - Iteração 5. Acha menor: [1, 2, 3, 5, 90, 6] Faz troca: [1, 2, 3, 5, 6, 90]



Selection Sort (Ordenação por Seleção)

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar o índice do menor elemento em uma lista, a partir de uma posição inicial:

```
menor = inicio
for j in range(inicio, fim):
    if lista[menor] > lista[j]:
        menor = j
```

Selection Sort (Ordenação por Seleção)

- Criamos então uma função que retorna o índice do elemento mínimo de uma lista, a partir de uma posição `inicio` passado por parâmetro:

```
def indiceMenor(lista, inicio):  
    menor = inicio  
    for j in range(inicio, len(lista)):  
        if lista[menor] > lista[j]:  
            menor = j  
    return menor
```

Selection Sort (Ordenação por Seleção)

- Dado a função anterior para achar o índice do menor elemento, como implementar o algoritmo de ordenação?
 - Ache o menor elemento a partir da posição 0, e troque com o elemento da posição 0.
 - Ache o menor elemento a partir da posição 1, e troque com o elemento da posição 1.
 - Ache o menor elemento a partir da posição 2, e troque com o elemento da posição 2.
 - E assim sucessivamente...

Selection Sort (Ordenação por Seleção)

```
def selectionSort(lista):  
    for i in range(len(lista)-1):  
        #Acha o menor elemento a partir da posição i  
        menor = indiceMenor(lista, i)  
        #Troca com o elemento da posição i  
        aux = lista[i]  
        lista[i] = lista[menor]  
        lista[menor] = aux
```

Selection Sort (Ordenação por Seleção)

```
lista = [14, 7, 8, 34, 56, 4, 0, 9, -8, 100]  
selectionSort(lista)  
lista
```

```
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
```

Selection Sort (Ordenação por Seleção)

- Passo a passo para [14, 7, 8, 34, 56, 4, 0, 9, -8, 100]:

```
[ -8, 7, 8, 34, 56, 4, 0, 9, 14, 100]
[-8, 0, 8, 34, 56, 4, 7, 9, 14, 100]
[-8, 0, 4, 34, 56, 8, 7, 9, 14, 100]
[-8, 0, 4, 7, 56, 8, 34, 9, 14, 100]
[-8, 0, 4, 7, 8, 56, 34, 9, 14, 100]
[-8, 0, 4, 7, 8, 9, 34, 56, 14, 100]
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
```

Selection Sort (Ordenação por Seleção)

- O uso da função para achar o índice do menor elemento não é estritamente necessária.

```
def selectionSort(lista):  
    for i in range(len(lista)-1):  
        #Acha o menor elemento a partir da posição i  
        menor = i  
        for j in range(i, len(lista)):  
            if lista[menor] > lista[j]:  
                menor = j  
        #Troca com o elemento da posição i  
        aux = lista[i]  
        lista[i] = lista[menor]  
        lista[menor] = aux
```

Selection Sort (Ordenação por Seleção)

- É muito comum a operação de troca de valores entre duas posições de uma lista.
- Python possui uma sintaxe resumida para fazer estas trocas.

```
def selectionSort(lista):  
    for i in range(len(lista)-1):  
        #Acha o menor elemento a partir da posição i  
        menor = i  
        for j in range(i, len(lista)):  
            if lista[menor] > lista[j]:  
                menor = j  
        #Troca com o elemento da posição i  
        lista[i], lista[menor] = lista[menor], lista[i]
```

Bubble Sort

(Ordenação por Bolha)

Bubble Sort (Ordenação por Bolha)

- Seja `lista` uma lista contendo números.
 - `lista = [5, 3, 2, 1, 90, 6]`.
- Seja `tam` o tamanho da lista.
- Devemos deixar `lista` em ordem crescente.

Bubble Sort (Ordenação por Bolha)

- A ideia do algoritmo é a seguinte:
 - Compare `lista[0]` com `lista[1]` e troque-os se `lista[0] > lista[1]`.
 - Compare `lista[1]` com `lista[2]` e troque-os se `lista[1] > lista[2]`.
 - Compare `lista[2]` com `lista[3]` e troque-os se `lista[2] > lista[3]`.
 - ...
 - Compare `lista[tam-2]` com `lista[tam-1]` e troque-os se `lista[tam-2] > lista[tam-1]`.
 - E assim sucessivamente ...

Bubble Sort (Ordenação por Bolha)

- Após uma iteração repetindo estes passos o que podemos garantir?
 - O maior elemento estará na posição correta!
- Após outra iteração de trocas, o segundo maior elemento estará na posição correta.
- E assim sucessivamente.
- Quantas iterações destas trocas precisamos para deixar a lista ordenada?

Bubble Sort (Ordenação por Bolha)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 1. [5, 3, 2, 1, 90, 6] Faz troca: [3, 5, 2, 1, 90, 6]
[3, 5, 2, 1, 90, 6] Faz troca: [3, 2, 5, 1, 90, 6]
[3, 2, 5, 1, 90, 6] Faz troca: [3, 2, 1, 5, 90, 6]
[3, 2, 1, 5, 90, 6]
[3, 2, 1, 5, 90, 6] Faz troca: [3, 2, 1, 5, 6, 90]
- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!

Bubble Sort (Ordenação por Bolha)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 2. [3, 2, 1, 5, 6, 90] Faz troca: [2, 3, 1, 5, 6, 90]
[2, 3, 1, 5, 6, 90] Faz troca: [2, 1, 3, 5, 6, 90]
[2, 1, 3, 5, 6, 90]
[2, 1, 3, 5, 6, 90]

Bubble Sort (Ordenação por Bolha)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - Iteração 3. [2, 1, 3, 5, 6, 90] Faz troca: [1, 2, 3, 5, 6, 90]
[1, 2, 3, 5, 6, 90]
[1, 2, 3, 5, 6, 90]
 - Iteração 4. [1, 2, 3, 5, 6, 90]
[1, 2, 3, 5, 6, 90]
 - Iteração 5. [1, 2, 3, 5, 6, 90]

Bubble Sort (Ordenação por Bolha)

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: 0 e 1; 1 e 2; ...; $i-1$ e i .
- Assumimos que de $(i+1)$ até $(tam-1)$, a lista já tem os maiores elementos ordenados.

```
for j in range(i):  
    if lista[j] > lista[j+1]:  
        lista[j], lista[j+1] = lista[j+1], lista[j]
```

Bubble Sort (Ordenação por Bolha)

```
def bubbleSort(lista):  
    #Índices i em ordem decrescente  
    for i in range(len(lista)-1,0,-1):  
        #Troca com o elemento da posição i  
        for j in range(i):  
            if lista[j] > lista[j+1]:  
                lista[j], lista[j+1] = lista[j+1], lista[j]
```

Bubble Sort (Ordenação por Bolha)

- Notem que as trocas na primeira iteração ocorrem até a última posição.
- Na segunda iteração ocorrem até a penúltima posição.
- E assim sucessivamente.
- Por que?

Insertion Sort

(Ordenação por Inserção)

Insertion Sort (Ordenação por Inserção)

- Seja `lista` uma lista contendo números.
 - `lista = [5, 3, 2, 1, 90, 6]`.
- Devemos deixar `lista` em ordem crescente.

Insertion Sort (Ordenação por Inserção)

- A ideia do algoritmo é a seguinte:
 - A cada passo, uma porção de 0 até $i-1$ da lista já está ordenada.
 - Devemos inserir o item da posição i na posição correta para deixar a lista ordenada até a posição i .
 - No passo seguinte consideramos que a lista está ordenado até i .

Insertion Sort (Ordenação por Inserção)

- Exemplo: [5, 3, 2, 1, 90, 6].
 - [5, 3, 2, 1, 90, 6] : lista ordenada de 0-0.
 - [3, 5, 2, 1, 90, 6] : lista ordenada de 0-1.
 - [2, 3, 5, 1, 90, 6] : lista ordenada de 0-2.
 - [1, 2, 3, 5, 90, 6] : lista ordenada de 0-3.
 - [1, 2, 3, 5, 90, 6] : lista ordenada de 0-4.
 - [1, 2, 3, 5, 6, 90] : lista ordenada de 0-5.

Insertion Sort (Ordenação por Inserção)

- Vamos supor que a lista está ordenada de 0 até $i-1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = lista[i]  #inserir aux na posição correta
j = i-1 #analisar elementos das posições anteriores
while (j >=0 and lista[j] > aux): #enquanto lista[j] > lista[i] empurra
    lista[j+1] = lista[j] #lista[j] para frente
    j = j-1
lista[j+1] = aux
```

Insertion Sort (Ordenação por Inserção)

- Exemplo $[1, 3, 5, 10, 20, 2, 4]$ com $i=5$.
 - $[1, 3, 5, 10, 20, 2, 4]$: $aux=2$, $j=4$.
 - $[1, 3, 5, 10, 20, 2, 4]$: $aux=2$, $j=3$.
 - $[1, 3, 5, 10, 20, 2, 4]$: $aux=2$, $j=2$.
 - $[1, 3, 5, 10, 20, 2, 4]$: $aux=2$, $j=1$.
 - $[1, 3, 5, 10, 20, 2, 4]$: $aux=2$, $j=0$.
- Aqui temos que $lista[j] < aux$ logo fazemos $lista[j+1] = aux$.
- $[1, 2, 3, 5, 10, 20, 4]$: $aux=2$, $j=0$.

Insertion Sort (Ordenação por Inserção)

```
def insertionSort(lista):  
    for i in range(1, len(lista)):  
        aux = lista[i]  
        j = i - 1  
        while (j >= 0 and lista[j] > aux): #põe elementos lista[j] > lista[i]  
            lista[j + 1] = lista[j] #para frente  
            j = j - 1  
        lista[j + 1] = aux #põe lista[i] na posição correta
```

Exercícios

Exercício

- Altere os algoritmos vistos nesta aula para que estes ordenem uma lista de inteiros em ordem **decrecente** ao invés de ordem crescente.

Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp).
- <https://panda.ime.usp.br/aulasPython/static/aulasPython/aula24.html>
- Selection Sort 3D Animation: <https://youtu.be/EdUWyka7kpl?t=57s>
- Bubble Sort 3D Animation: <https://youtu.be/NiyEqLZmngY?t=55s>