MC-102 — Aula 21
Ordenação – SelectionSort e BubbleSort

Instituto de Computação – Unicamp

14 de Maio de 2018

Roteiro

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.

- Nos nossos exemplos usaremos um lista de números como a coleção.
  - É claro que quaisquer números possuem uma relação de ordem entre si.
- Apesar de usarmos inteiros, os algoritmos servem para ordenar qualquer coleção de elementos que possam ser comparados.

Ordenação

- O problema de ordenação é um dos mais básicos em computação.
  - Mas muito provavelmente é um dos problemas com o maior número de aplicações diretas ou indiretas (como parte da solução para um problema maior).
- Exemplos de aplicações diretas:
  - Criação de *rankings*, Definir preferências em atendimentos por prioridade, Criação de Listas etc.
- Exemplos de aplicações indiretas:
  - Otimizar sistemas de busca, manutenção de estruturas de bancos de dados etc.

Selection-Sort

- Seja **vet** uma lista contendo números.
- Devemos deixar **vet** em ordem crescente.
- A idéia 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

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

Iteração 0. Acha menor: [5,3,2,1,90,6]. Faz troca: [1,3,2,5,90,6].

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

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

Selection-Sort

Exemplo: [5,3,2,1,90,6].
Iteração 0. Acha menor: [5,3,2,1,90,6]. Faz troca: [1,3,2,5,90,6].
Iteração 1. Acha menor: [1,3,2,5,90,6]. Faz troca: [1,2,3,5,90,6].
Iteração 2. Acha menor: [1,2,3,5,90,6]. 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 5: Acha menor: [1,2,3,5,90,6]. Faz troca: [1,2,3,5,6,90].

Selection-Sort

Exemplo: [5,3,2,1,90,6].
Iteração 0. Acha menor: [5,3,2,1,90,6]. Faz troca: [1,3,2,5,90,6].
Iteração 1. Acha menor: [1,3,2,5,90,6]. Faz troca: [1,2,3,5,90,6].
Iteração 2. Acha menor: [1,2,3,5,90,6]. 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 5: Acha menor: [1,2,3,5,90,6]. Faz troca: [1,2,3,5,6,90].

Selection-Sort

Exemplo: [5,3,2,1,90,6].
Iteração 0. Acha menor: [5,3,2,1,90,6]. Faz troca: [1,3,2,5,90,6].
Iteração 1. Acha menor: [1,3,2,5,90,6]. Faz troca: [1,2,3,5,90,6].
Iteração 2. Acha menor: [1,2,3,5,90,6]. 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 5: Acha menor: [1,2,3,5,90,6]. Faz troca: [1,2,3,5,6,90].

Selection-Sort

Exemplo: [5,3,2,1,90,6].
Iteração 0. Acha menor: [5,3,2,1,90,6]. Faz troca: [1,3,2,5,90,6].
Iteração 1. Acha menor: [1,3,2,5,90,6]. Faz troca: [1,2,3,5,90,6].
Iteração 2. Acha menor: [1,2,3,5,90,6]. 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 5: Acha menor: [1,2,3,5,90,6]. Faz troca: [1,2,3,5,6,90].

Selection-Sort

- 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:

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

Selection-Sort

- 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:

```
min = inicio
for i in range(inicio,fim):
  if vet[min] > vet[j]:
    min = j
```

Selection-Sort

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

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

Selection-Sort

- 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

```
def selectionSort(vet):
    for i in range(len(vet)-1):
        min = indiceMenor(vet, i)
        aux = vet[i]
        vet[i] = vet[min]
        vet[min] = aux
```

Selection-Sort

```
>> 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

- O uso da função para achar o índice do menor elemento não é estritamente necessária.
- Podemos refazer a função selectionSort como segue:

```
def selectionSort2(vet):
  for i in range(len(vet)-1):
    min = i
    for j in range(i,len(vet)):
      if vet[min] > vet[j]:
        min = j

    aux = vet[i]
    vet[i] = vet[min]
    vet[min] = aux
```

Selection-Sort

- É muito comum a operação de troca de valores entre duas posições de uma lista.
- Python possui uma sintaxe resumida para fazer estas trocas.
- Veja o uso no algoritmo do slide anterior.

```
def selectionSort2(vet):
  for i in range(len(vet)-1):
    min = i
    for j in range(i,len(vet)):
      if vet[min] > vet[j]:
        min = j

    vet[i], vet[min] = vet[min], vet[i]
```

Bubble-Sort

- Seja **vet** um lista contendo números. *tam* é tamanho da lista.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Seja **vet** um lista contendo números. *tam* é tamanho da lista.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Após uma iteração de trocas, o maior elemento estará na última posição.
- Após outra iteração de trocas, o segundo maior elemento estará na posição correta.
- E assim sucessivamente.
- Quantas iterações destas trocas precisamos para deixar o vetor ordenado?

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:

$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: [5,3,2,1,90,6].
Valores sublinhados estão sendo comparados:
$[\underline{5}, \underline{3}, 2, 1, 90, 6]$
$[3, \underline{5}, \underline{2}, 1, 90, 6]$
$[3, 2, \underline{5}, \underline{1}, 90, 6]$
$[3, 2, 1, \underline{5}, \underline{90}, 6]$
$[3, 2, 1, 5, \underline{90}, \underline{6}]$
$[3, 2, 1, 5, 6, 90]$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

- 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 $(tam-1)$, o vetor já tem os maiores elementos ordenados.

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

Bubble-Sort

- 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 $(tam-1)$, o vetor já tem os maiores elementos ordenados.

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

Bubble-Sort

```
def bubbleSort (vet):
    for i in range(len(vet)-1,0,-1): #índices i em ordem decrescente
        for j in range(i):
            if vet[j] > vet[j+1] :
                vet[j], vet[j+1] = vet[j+1], vet[j]
```

Bubble-Sort

- 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?

Exercício

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

Exercício

No algoritmo SelectionSort, o for do i vai de 0 até $len(vet) - 2$. (o range é $range(0, len(vet) - 1)$ o que significa que o i vai até $len(vet) - 2$.) Por que não ir até $len(vet) - 1$?

```
def selectionSort2(vet):
  for i in range(len(vet)-1): #Por que não usar range(len(vet))?
    min = i
    for j in range(i,len(vet)):
      if vet[min] > vet[j]:
        min = j

    vet[i], vet[min] = vet[min], vet[i]
```