16 Aula 16: 03/OUT/2019

16.1 Aula passada

Problema: rearranjar os elementos de uma lista para ficarem em ordem crescente.

Os algoritmos para esse problema estão servindo como laboratório de ideias para projeto e análise de algoritmos.

Na aula passada vimos ordenação por inserção, ordenação por inserção binária. Fizemos a análise desses algoritmo e vimos notação assintótica. Ordenação por seleção é um exemplo de algoritmo dinâmico: os dados não precisam estar todos presentes no início, podem ser fornecido *online*.

algoritmo	melhor caso	pior caso
inserção	$O(n)$	$O(n^2)$
inserção binária	$O(n \lg n)$	$O(n^2)$

Aproveitamos para mostrar a página *TimeComplexity* e o consumo de tempo dos métodos e funções do Python em notação assintótica.

16.2 Hoje

Nessa aula vimos um pouco mais de análise assintótica e experimental. Vimos ordenação por seleção e algumas versões do `bubblesort()` e algumas ideias de melhorias.

16.3 Arquivos com material complementar

No diretório da aula há alguns slides que podem ser úteis para simulações:

- `slides_analise_experimental.pdf`: tabela com análie experimental dos algoritmos de ordenação quadráticos
- `slides_notacao_assintotica.pdf`: define a classe O grande; possui uma tabela comparando funções e comentando sobre a relação de consumo de tempo dessas funções. Tem as classes de complexidade: linear, quadrática, $n \lg n$, cúbica

16.4 Ordenação por seleçao

Há duas versões, pegue o menor e mova para o início ou pegue o maior mova para o final.

```
#--------------------------------------
def selecao(lista):
    '''(list) -> None '''
    n = len(lista)
    i = 0
    while i < n-1:
        # encontra o dono da posição i
        i_min = i
```

```
        j = i+1
        while j < n:
            if lista[j] < lista[i_min]:
                i_min = j
            j += 1
        # troca
        lista[i],lista[i_min] = lista[i_min], lista[i]
        i += 1

#-------------------------------------------
def selecao(lista):
    '''(list) -> None '''
    n = len(lista)
    for i in range(n-1):
        # encontra o dono da posição i
        i_min = i
        for j in range(i+1,n):
            if lista[j] < lista[i_min]:
                i_min = j
        # troca
        lista[i],lista[i_min] = lista[i_min], lista[i]
```

16.4.1 Invariantes da ordenação por seleção

- a lista é uma permutação da original
- na linha do `while i < n-1:` vale que `lista[0:i]` está em ordem crescente
- na linha do `while i < n-1:` vale que `lista[0:i] <= lista[i:n]` está em ordem crescente

16.4.2 Ordenação por seleção pythoniana

Consumo de tempo assintótico é o mesmo. Mais eficiente na prática

```
#-------------------------------------------------------------
def selecao_py (v, e, d):
    '''(list,int,int) -> None

    Recebe uma lista v[e:d] e ordena os seus elementos
    em ordem crescente.

    A função implementa o algoritmo de ordenação por seleção e usa
    funções nativas.
    '''
    for i in range(e, d):
        clone = v[i: d]
        mini = min(clone)   # indice do candidato a menor item de v[i:d]
        imin = clone.index(mini)
        # troca v[i] e v[imin]
        v[imin], v[i] = v[i], v[imin]
```

16.4.3 Análise experimental

A análise experimental comprova que não importa a implementação, usando ou não funções nativas, o consumo é $O(n^2)$

```
aulas/sort> python main.py -s -p
    n     selecao selecao_P
    256      0.00      0.00
    512      0.01      0.01
   1024      0.05      0.02
   2048      0.23      0.09
   4096      1.00      0.40
   8192      4.34      1.59
  16384     18.54      6.91
  ...
```

16.5 Bubblesort

```
    v     n=12
    +----+----+----+----+----+----+----+----+----+----+----+----+----+
    | 50 | 35 | 99 | 38 | 55 | 20 | 44 | 10 | 40 | 65 | 25 | 35 |    |
    +----+----+----+----+----+----+----+----+----+----+----+----+----+
       0    1    2    3    4    5    6    7    8    9   10   11   12

    50                          10 |
    35                       10 50 |
    99                    10 35    |
    38                 10 99       |
    55              10 38          |
    20           10 55             |
    44        10 20                |
    10        44                   |
    40    25                       |
    65 25 40                       |
    25 65                          |
    35                             |
```

```
#---------------------------------------
def bubble_sort(lista):
    n = len(lista)
    i = 0
    while i < n-1:
        j = n-1
        while j > i:
            if lista[j] < lista[j-1]:
                # troca
                aux = lista[j]
                lista[j] = lista[j-1]
                lista[j-1] = aux
            j-=1
        i+=1


#---------------------------------------
def bubble_sort(lista):
    n = len(lista)
    for i in range(n-1):
        for j in range(n-1,i,-1):
            if lista[j] < lista[j-1]:
                # troca
                aux = lista[j]
                lista[j] = lista[j-1]
                lista[j-1] = aux

#---------------------------------------
```

```
def bubble_sort(lista):
    n = len(lista)
    i = 0
    while i < n-1:
        ult_troca = i
        j = n-1
        while j > i:
            if lista[j] < lista[j-1]:
                # troca
                aux = lista[j]
                lista[j] = lista[j-1]
                lista[j-1] = aux
                ult_troca = j
            j-=1
        i = ult_troca + 1



#-------------------------------------------------------------
def bubble_sort(v):
    '''(list) -> None

    Recebe uma lista v e ordena os seus elementos
    em ordem crescente.

    A função implementa o buble sort que também é
    conhecido como ordenação por troca,
    '''
    n = len(v)
    for i in range(n):
        for j in range(n-1,i,-1):
            if v[j] < v[j-1]:
                # troca
                v[j], v[j-1] = v[j-1], v[j]


#-------------------------------------------------------------
def bubble_sort2(v):
    '''(list) -> None

    Recebe uma lista v e ordena os seus elementos
    em ordem crescente.

    A função implementa o bubble sort que também é
    conhecido como ordenação por troca.

    Utiliza marcador de última troca.
    '''
    n = len(v)
    i = 0
```

```
    while i < n:
        ult_troca = n # posicao da ultima troca
        for j in range(n-1,i,-1):
            if v[j] < v[j-1]:
                # troca v[j] e v[j-1]
                v[j], v[j-1] = v[j-1], v[j]

                # guarda a posicao onde ocorreu a ultima troca
                ult_troca = j

        i = ult_troca


#-------------------------------------------------------------
def shaker(v):
    '''(list) -> None
    Rearranja os elementos de `v`de tal forma que fique crescente.
    '''
    n = len(v)
    ini = 0
    fim = n
    # relações invariantes v[0:ini] e v[fim:n] são crescentes
    # para x em v[0:ini] e y em v[ini:n] vale que x <= y
    # para x em v[0:fim] e y em v[fim:n] vale que x <= y

    while ini < fim:
        # mais leve sobe...
        ult_troca = fim
        for j in range(fim-1,ini,-1):
            if v[j] < v[j-1]:
                v[j], v[j-1] = v[j-1], v[j]
                # guarda a posicao onde ocorreu a ultima troca
                ult_troca = j
        ini = ult_troca

        # mais pesado desce...
        for j in range(ini,fim-1,+1):
            if v[j] > v[j+1]:
                v[j], v[j+1] = v[j+1], v[j]
                # guarda a posicao onde ocorreu a ultima troca
                ult_troca = j+1
        fim = ult_troca
```

16.5.1 Invariantes do bubblesort

- a lista é uma permutação da original
- na linha do `while i < n-1:` vale que `lista[0:i]` está em ordem crescente
- na linha do `while i < n-1:` vale que `lista[0:i] <= lista[i:n]`