15 Aula 15: 01/OUT/2019

15.1 Aulas passadas

- recursão: várias
- `FibonacciR()`: com *memoization* para "encerrar" recursão.
- *busca sequencial*: mostra começar a contar operações e estimativa de consumo de tempo
- *busca binárias*: motra um algoritmo de busca muito eficiente para objetos ordenados. Algoritmo recursivo que podemos escrever versões recursivas e iterativas.

15.2 Hoje

Rudimentos de análise experimental e analítica de algoritmos.

Utilizaremos como laboratório de ideias o **problema de ordenação**. Esse problema nos levará a muitas técnicas em análise e projeto de algoritmos que serão vistos nas próximas aulas. Veremos **notação assintótica** e a página *TimeComplexity* (link).

Na prática, para ordenar uma lista, temos o método nativo `list.sort()` que é muito eficiente.

15.3 Material complementar

O diretório da aula contém slides que podem ser úteis para simulações:

- `slides_ordenacao_insercao.pdf`: sobre ordenação por inserção, há uma tabela que compara os valores das funções $n^2 + 3n - 2$ e que mostra que quando n cresce a contribuição de $3n - 2$ passa a ser desprezível. Isso pode nos levar a notação assintótica.
- `slides_notacao_assintotica.pdf`: define a classe Oh grande; possui uma tabela comparando funções e comentando sobre a relação de consumo de tempo dessas funções.
- `slides_ordenacao_insercao_ninaria.pdf`: trata sobre *ordenação por inserção binária* que, apesar de não melhor a ordenação do ponto e vista assintótico é mais eficiente na prática.

15.4 Algoritmos básicos de ordenação

15.4.1 Problema 1: inserir um elemento em lista ordenada

Escreva um trecho de código que recebe uma lista `v[0:i+1]` tal que `v[0:i]` é crescente e rearranja seus elementos de maneira que `v[0:i+1]` seja crescente.

```
# solução 1
pivo = v[i]
j = i-1
while j >= 0 and v[j] > pivo:
    v[j+1] = v[j]
    j -= 1
v[j+1] = pivo
```

```
# solução 2
pivo = v[i]
j = i-1
# encontra a posição j+1 onde pivo deve ser inserido
while j >= 0 and v[j] > pivo:
    j -= 1
# desloca os itens
for k in range(i,j+1,-1):
    v[k] = v[k-1]
# insere o elemento
v[j+1] = pivo
```

15.4.2 Problema 2: ordenação

Usando o trecho de código anterior, escreva uma função que recebe uma lista `v` e rearranja seus `n` elementos de tal forma que a lista fique em ordem crescente.

15.4.2.1 Solução: ordenação de inserção

```
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
```

Simular.

```python
#----------------------------------------
def insercao(v):
1    n = len(v)
2    for i in range(1, n):
3        pivo = v[i]
         # encontre posição j tal que v[j] <= pivo < v[j+1]
4        j = i-1
5        while j >= 0 and v[j] > pivo:
6            j -= 1
         # insira o elemento na posição j+1
7        for k in range(i, j+1, -1):
8            v[k] = v[k-1]
         # coloque o pivo na sua posição
9        v[j+1] = pivo
```

```python
def insercao(v):
    n = len(v)
    for i in range(1, n):
        pivo = v[i]
        j = i-1
        while j >= 0 and v[j] > pivo:
            v[j+1] = v[j]
            j -= 1
        v[j+1] = pivo
```

15.4.2.2 Invariantes da ordenação por inserção

- a lista é uma permutação da original
- na linha do `for i in range(1,n):` vale que `v[0:i]` é crescente

15.4.2.3 Consumo de tempo

```
linha    número de execuções da linha
  1         = 1
  2         = n
  3-4       = n-1
  5-6       <= 1 + 2 + ... + n-1 = (n*(n-1))/2
  7-8       <= (n*(n-1))/2
  9         = n-1
------------------------------------------------
Total       = n^2 + 2n -1  = O(n^2)
```

O consumo de tempo da ordenação inserção no melhor caso é proporcional a `n`.

O consumo de tempo da ordenação por inserção no pior caso é proporcional a n^2.

O consumo de tempo da ordenação por inserção é $O(n^2)$.

15.5 Notação assintótica

Sejam $\mathtt{T}(\mathtt{n})$ e $\mathtt{f}(\mathtt{n})$ funções dos inteiros nos reais.

Dizemos que $\mathtt{T}(\mathtt{n})$ é $\mathtt{O}(\mathtt{f}(\mathtt{n}))$ se existem constantes positivas $\mathtt{c}$ e $\mathtt{n}_0$ tais que

$$\mathtt{T}(\mathtt{n}) \ \leq \ \mathtt{c}\,\mathtt{f}(\mathtt{n})$$

para todo $\mathtt{n} \geq \mathtt{n}_0$.

15.5.1 Mais informal

$\mathtt{T}(\mathtt{n})$ é $\mathtt{O}(\mathtt{f}(\mathtt{n}))$ se existe $\mathtt{c} > \mathtt{0}$ tal que $\mathtt{T}(\mathtt{n}) \ \leq \ \mathtt{cf}(\mathtt{n})$, para todo n *suficientemente* GRANDE.

15.6 Ordenação por inserção binária

Ideia: trocar a busca sequencial por binária.

```
def insercao(v):
1     n = len(v)
2     for i in range(1, n):
3         pivo = v[i]
          # encontre posição j tal que v[j] <= pivo < v[j+1]
4         j = i-1
5         while j >= 0 and v[j] > pivo:
6             j -= 1
          # insira o elemento na posição j+1
7         for k in range(i, j+1, -1):
8            v[k] = v[k-1]
          # coloque o pivo na sua posição
9         v[j+1] = pivo

def insercao_binaria(v):
1     n = len(v)
2     for i in range(1, n):
3         pivo = v[i]
          # encontre posição j tal que v[j] <= pivo < v[j+1]
5-6       j = busca_binaria(x, i, v)
          # insira o elemento na posição j+1
7         for k in range(i, j+1, -1):
8             v[k] = v[k-1]
          # coloque o pivo na sua posição
9         v[j+1] = pivo


def busca_binaria(x, n, v):
    '''(valor, list) -> int

    Recebe uma v crescente v[0:n] com n >= 1
    e um valor x e retorna um índice j em [0:n]
    tal que v[j] <= x < v[j+1]
    '''
    e, d = 0, n
    while e < d:
        m = (e + d) // 2
        if v[m] <= x: e = m+1
        else: d = m
    return e
```

15.6.1 Consumo de tempo

```
linha    número de execuções da linha
  1          = 1
```

```
  2          = n
  3-4        = n-1
  5-6        <= lg 1 + lg 2 + ... + lg (n-1) <= n lg n
  7-8        <= (n*(n-1))/2
  9          = n-1
------------------------------------------------
Total        = n^2/2 + n lg n + 7n/2 - 1  = O(n^2)
```

15.6.2 Consumo de tempo

O consumo de tempo da inserção binária no melhor caso é proporcional a $\mathtt{n} \lg \mathtt{n}$.

O consumo de tempo da inserção binária no pior caso é proporcional a $\mathtt{n}^2$.

O consumo de tempo da inserção binária é $\mathtt{O}(\mathtt{n}^2)$.