# Estrutura de dados
## Árvores
## parte 2

Profª. Drª. Alana Morais

# Recapitulando - Estrutura de dados

**Dados simples:**

- padrão:
  - inteiro (int);
  - real (float);
  - caracter (str);
  - lógico (boolean).

**Dados estruturados:**

- Estáticos:
  - arrays;
  - registros;
  - arquivos;
  - conjuntos;
  - cadeias.
- Dinâmicos:
  - filas;
  - pilhas;
  - listas encadeadas;
  - árvores;
  - grafos.

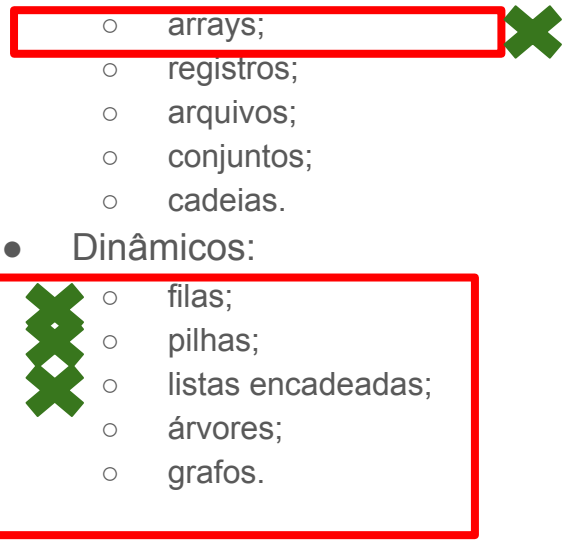
# Recapitulando - Estrutura de dados

**Dados simples:**

- padrão:
  - inteiro (int);
  - real (float);
  - caracter (str);
  - lógico (boolean).

**Dados estruturados:**

- Estáticos:
  - arrays;
  - registros;
  - arquivos;
  - conjuntos;
  - cadeias.
- Dinâmicos:
  - filas;
  - pilhas;
  - listas encadeadas;
  - árvores;
  - grafos.



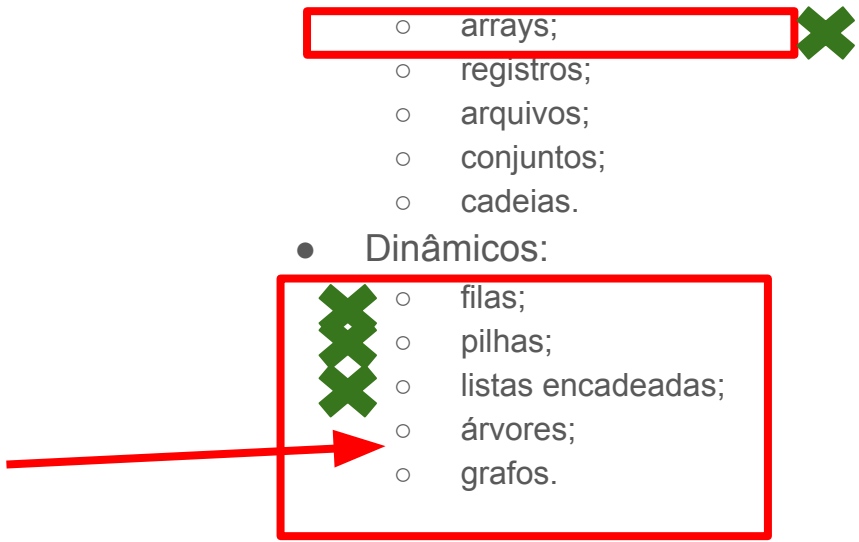
# Recapitulando - Estrutura de dados

**Dados simples:**

- padrão:
  - inteiro (int);
  - real (float);
  - caracter (str);
  - lógico (boolean).

**Dados estruturados:**

- Estáticos:
  - arrays;
  - registros;
  - arquivos;
  - conjuntos;
  - cadeias.
- Dinâmicos:
  - filas;
  - pilhas;
  - listas encadeadas;
  - árvores;
  - grafos.



# Operações com árvores

- ~~Construindo uma árvore;~~
- ~~Inserindo ordenado;~~
- ~~Buscando elementos na árvore;~~
- ~~Calculando a altura de uma árvore;~~
- Percorrendo e Mostrando;
- Exclusão de elementos

# Percorrendo uma Árvore

- Este é o processo de visitar cada nó da árvore exatamente uma vez.
- O percurso pode ser interpretado como colocar todos os nós em uma linha ou a linearização de uma árvore.
- A forma mais natural de percorrer uma árvore é fazer o percurso recursivamente.
- Por exemplo, se a árvore contém inteiros na carga, a função abaixo retorna a soma das cargas:

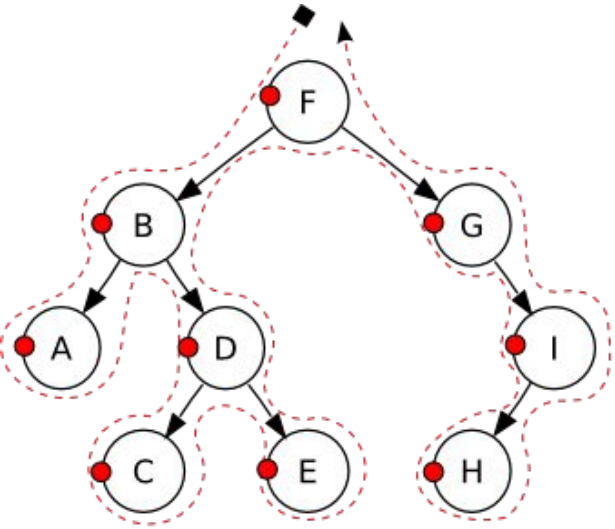
# Percorrendo uma Árvore

- 3 maneiras diferentes:
  pré-ordem
  chave -> esquerda -> direita

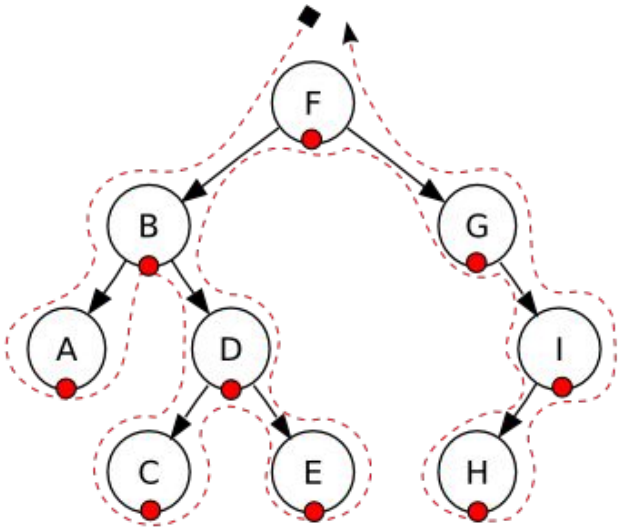
  em-Ordem
  esquerda -> chave -> direita

  pós-ordem
  esquerda -> direita -> chave

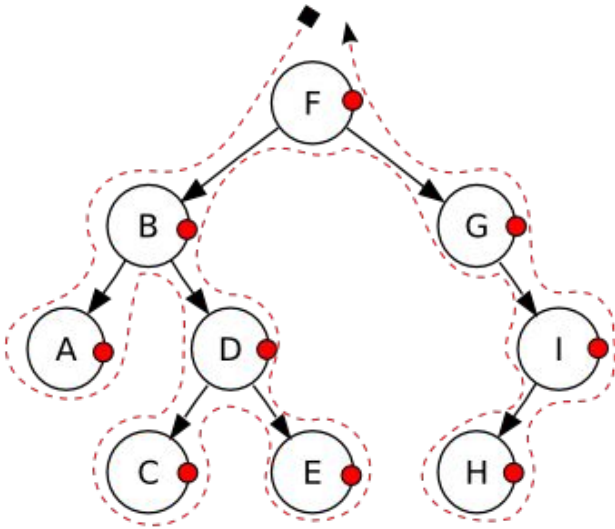
# Percorrendo uma Árvore




**Pré-ordem:** F, B, A, D, C, E, G, I, H




**Ordem simétrica:** A, B, C, D, E, F, G, H, I



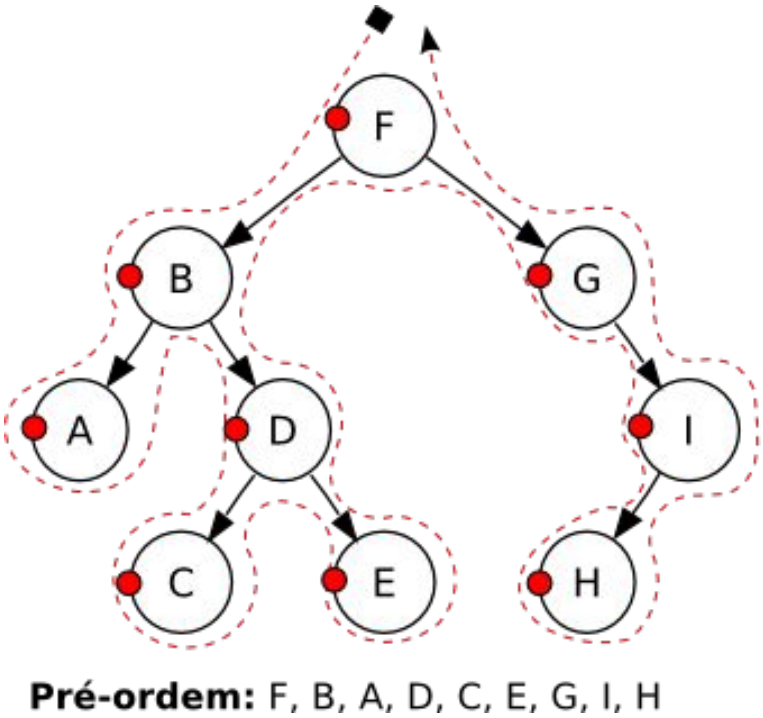

**Pós-ordem:** A, C, E, D, B, H, I, G, F

# Percorrendo uma Árvore … pré-ordem

tree.py

```
class Arvore:
     ...

def preOrdem(no):
    global ImprimeArvore
    if no is None:
        return
    ImprimeArvore += str(no.chave) + ', '
    preOrdem(no.esquerda)
    preOrdem(no.direita)
```

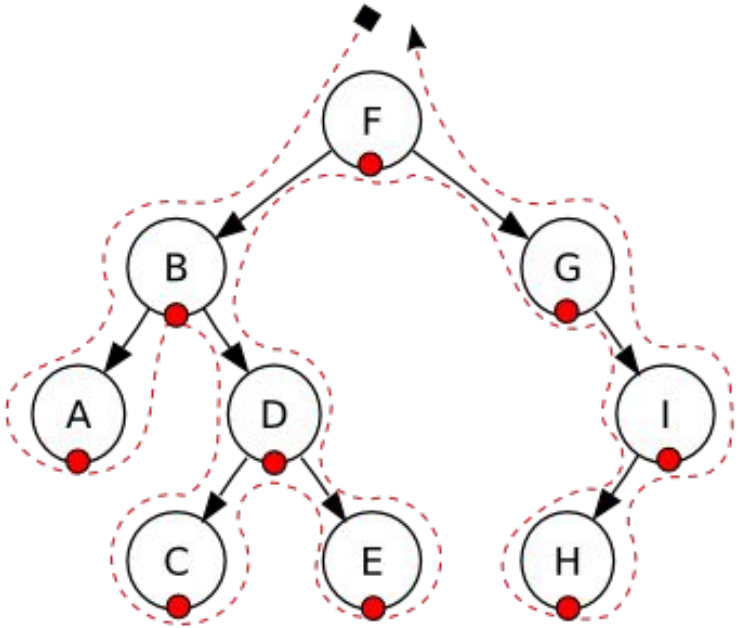


# Percorrendo uma Árvore … em ordem

tree.py

```
class Arvore:
     ...

def emOrdem(no):
    global ImprimeArvore
    if no is None:
        return
    emOrdem(no.esquerda)
    ImprimeArvore += str(no.chave) + ', '
    emOrdem(no.direita)
```



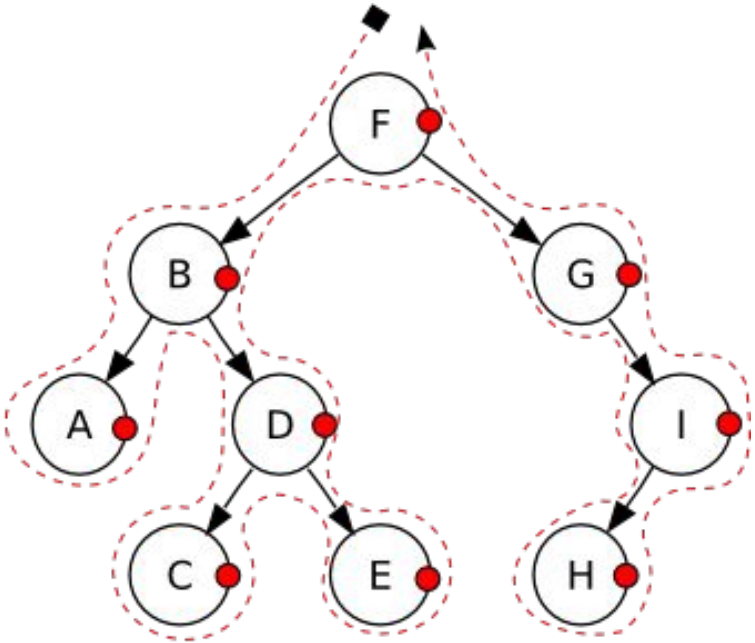

**Ordem simétrica:** A, B, C, D, E, F, G, H, I

# Percorrendo uma Árvore … pós-ordem

tree.py

```
class Arvore:
     ...

def posOrdem(no):
    global ImprimeArvore
    if no is None:
        return
    posOrdem(no.esquerda)
    posOrdem(no.direita)
    ImprimeArvore += str(no.chave) + ', '
```




**Pós-ordem:** A, C, E, D, B, H, I, G, F

# Percorrendo uma Árvore … pós-ordem

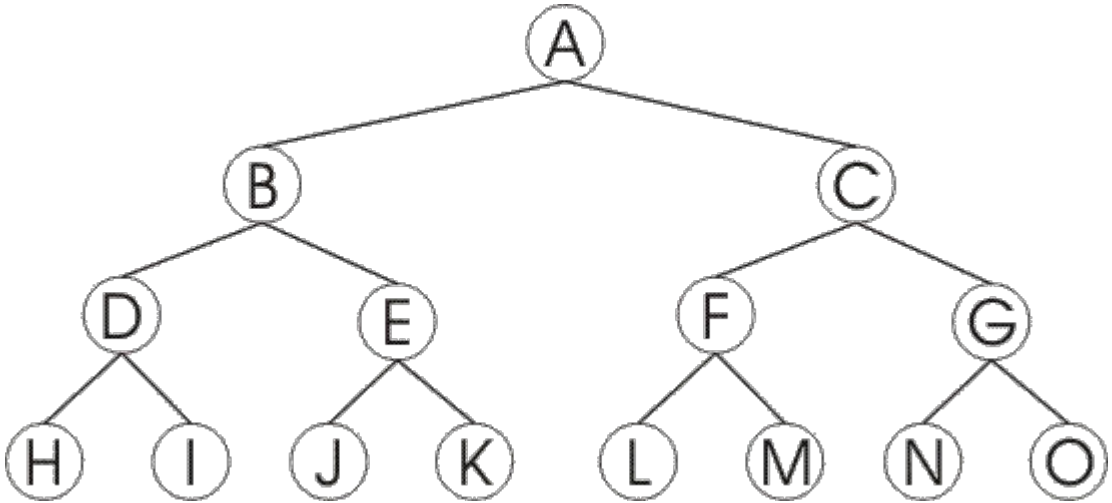
tree.py

```
class Arvore:
     ...

# Chama os metodos de impressao
ImprimeArvore = ""
preOrdem(arvore)
print "PreOrdem: " + ImprimeArvore + "\n"
ImprimeArvore = ""
emOrdem(arvore)
print "EmOrdem: " + ImprimeArvore + "\n"
ImprimeArvore = ""
posOrdem(arvore)
print "PosOrdem: " + ImprimeArvore + "\n"
```

# Exercício

Crie a árvore a seguir e a percorra de acordo com os algoritmos pré-ordem, em ordem e pós-ordem.



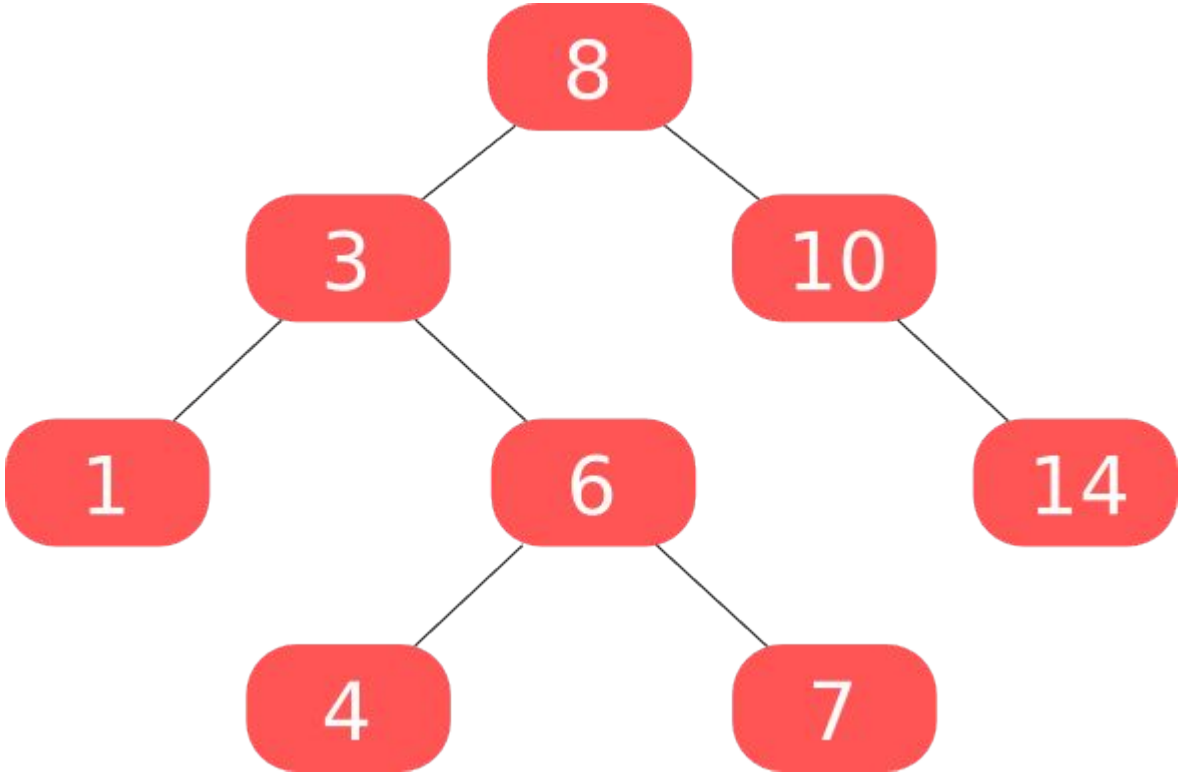
# Operações com árvores

- ~~Construindo uma árvore;~~
- ~~Inserindo ordenado;~~
- ~~Buscando elementos na árvore;~~
- ~~Calculando a altura de uma árvore;~~
- Percorrendo e Mostrando;
- Exclusão de elementos

# Excluindo elementos de uma Árvore

- A remoção de um elemento é um pouco mais complicada do que a inserção e busca de um elemento.
- Existem 3 situações diferentes e que requerem diferentes abordagens para a remoção de um elemento:
    a. o nó a ser removido é um nó folha
    b. o nó a ser removido possui somente um filho
    c. o nó a ser removido possui dois filhos

# Excluindo elementos de uma Árvore



# Excluindo elementos de uma Árvore

tree.py

```
class Arvore:
     ...
def buscaNoPai(no, ch):
    noPai = no
    while no is not None:
        if no.chave == ch:
            return noPai
        noPai = no
        if no.chave < ch:
            no = no.direita
        else:
            no = no.esquerda
    return noPai
```

# Excluindo elementos de uma Árvore

tree.py

```
class Arvore:
     ...

def maiorAesquerda(no):
    no = no.esquerda
    while no.direita is not None:
        no = no.direita
    return no
```

tree.py

```
class Arvore:
        ...
def exclui(no, ch):
    atual = buscaLinear(no, ch)
    if atual is None:
        return False
    noPai = buscaNoPai(no, ch)
    if atual.esquerda is None or atual.direita is None:
        if atual.esquerda is None:
            substituto = atual.direita
        else:
            substituto = atual.esquerda
        if noPai is None:
            no = substituto
        elif ch > noPai.chave:
            noPai.direita = substituto
        else:
            noPai.esquerda = substituto
    else:
        substituto = maiorAesquerda(atual)
        atual.chave = substituto.chave
        if substituto.esquerda is not None:
            atual.esquerda = substituto.esquerda
        else:
            atual.esquerda = None
    return True
```

# Excluindo elementos de uma Árvore

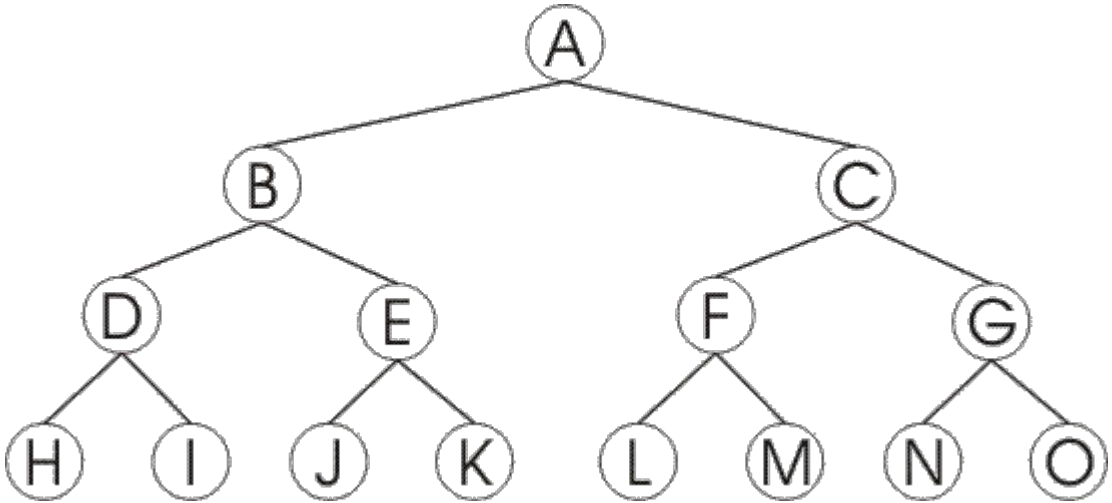
tree.py

```
class Arvore:
    …


exclui(arvore, 7)
exclui(arvore, 5)
exclui(arvore, 8)
exclui(arvore, 3)
```

# Exercício

Agora remova os elementos I, M e N.



# Exercício Final

Escreva funções:

a. para contar o número de nós em uma árvore binária
b. para contar o número de folhas
c. para contar o número de filhos à direita
d. para excluir todas as folhas de uma árvore binária
e. para testar se uma árvore binária é do tipo de busca

# Exercício

1. Implemente uma árvore binária de busca, crie uma função que verifique o menor elemento existente na mesma.

2. Implemente uma árvore binária de busca, crie uma função que verifique o maior elemento existente na mesma.

3. Verifique se duas árvores binárias são idênticas.

# Dúvidas?

alanamm.prof@gmail.com