# ÁRVORES BINÁRIAS

Python

# ÁRVORES BINÁRIAS

Características:

- utilizada em muitas aplicações;
- modela uma hierarquia de elementos;
  - árvore genealógica;
  - diagrama organizacional;
  - modelagem de algoritmo;
- trata-se de um conjunto finito de dados;
- é uma extensão de uma lista ligada

# ÁRVORES BINÁRIAS

Características:

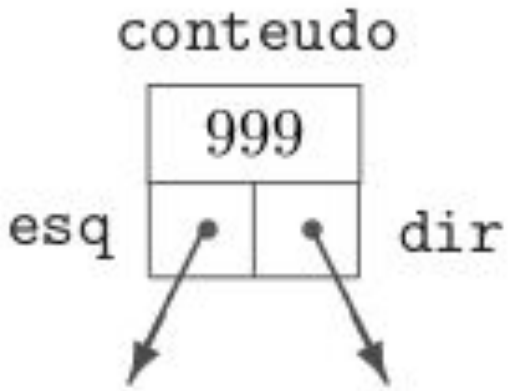
- conjunto de registros que satisfaz determinadas condições;
- estruturas de dados não linear;
- bidimensional, ou seja, cada nó tem apenas dois filhos;
- dados armazenados de forma hierárquica;
- cada registro é chamado de nó;
- cada nó possui um endereço;
- cada nó terá três campos:
  - valor
  - endereço do nó esquerdo (ponteiro);
  - endereço do nó direito (ponteiro);
- O número de campos em um nó pode variar em função da forma de navegação desejada;

# ÁRVORES BINÁRIAS

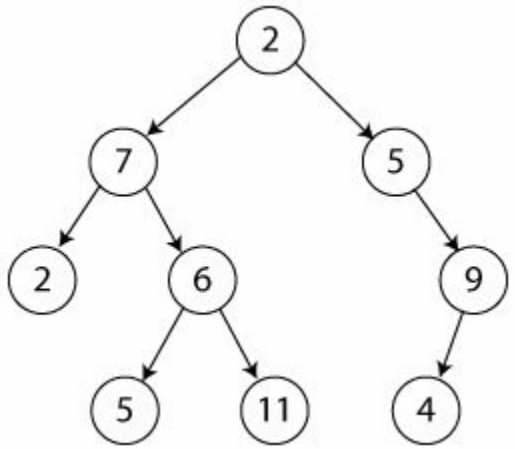
Características:

- definição de um nó:

```
typedef struct reg {
    int  conteudo; // conteúdo
    noh *esq;
    noh *dir;
 } noh; // nó
```





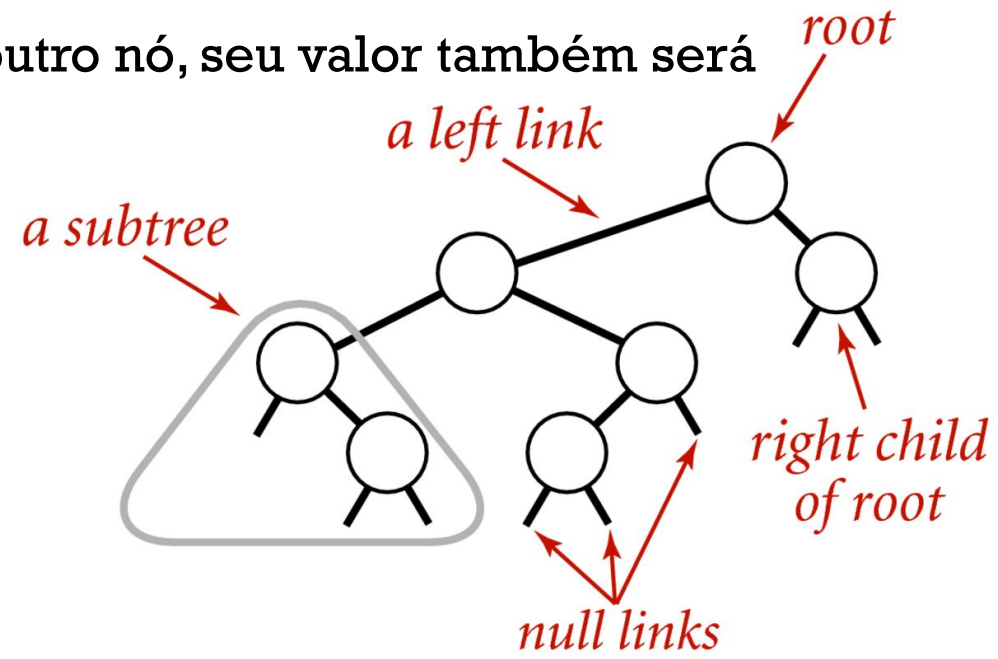


# ÁRVORES BINÁRIAS

Características:

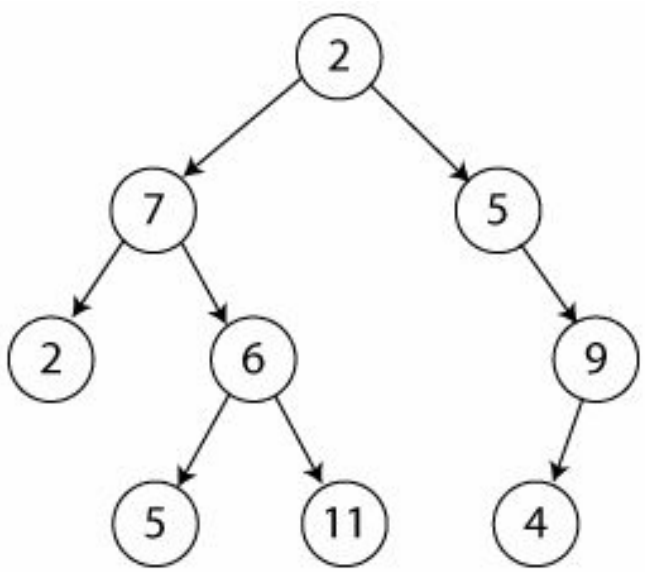
- o elemento inicial é chamado de raiz (root);
- pode ter nenhuma ou várias sub-árvores, que, por sua vez, se dividem a, subárvore esquerda e sub-árvore direita;
- uso do conceito de nó pai e nós filhos;
- caso esse nó seja raiz, o ponteiro do nó pai é igual a Nil.
- caso algum dos nós filhos não aponte para algum outro nó, seu valor também será Nil.
- esse nó, assim como nas listas encadeadas,

  é representado por um objeto.



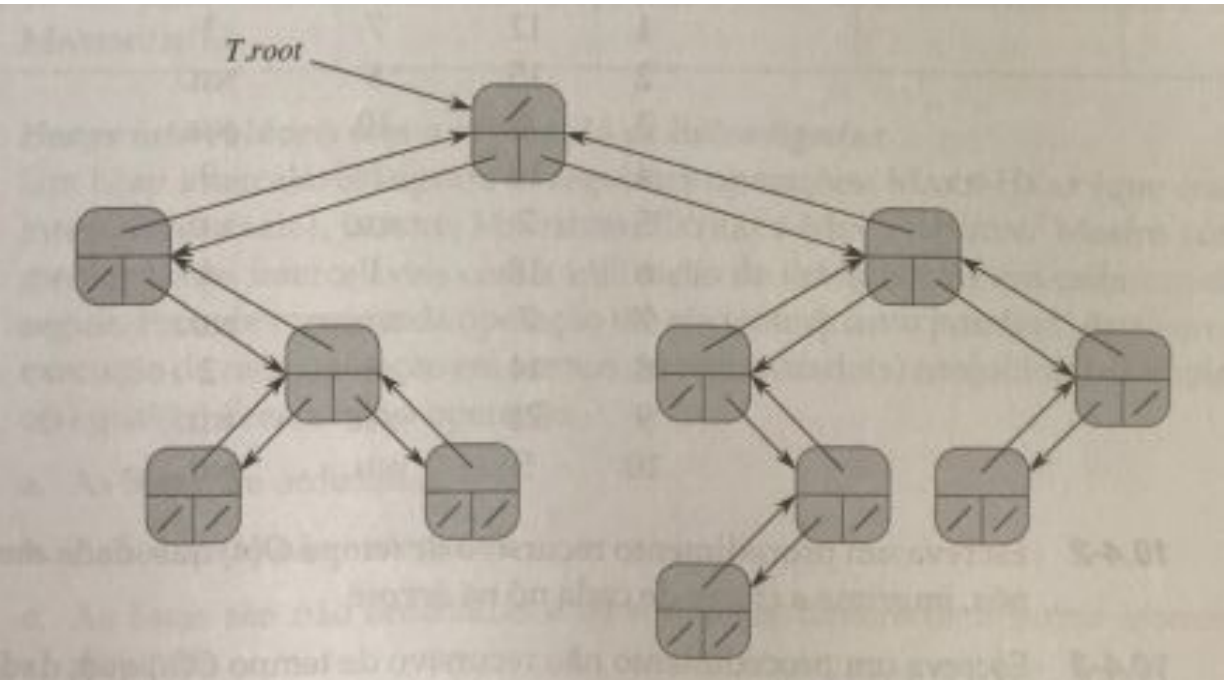
# ÁRVORES BINÁRIAS

Conceitos relacionados:

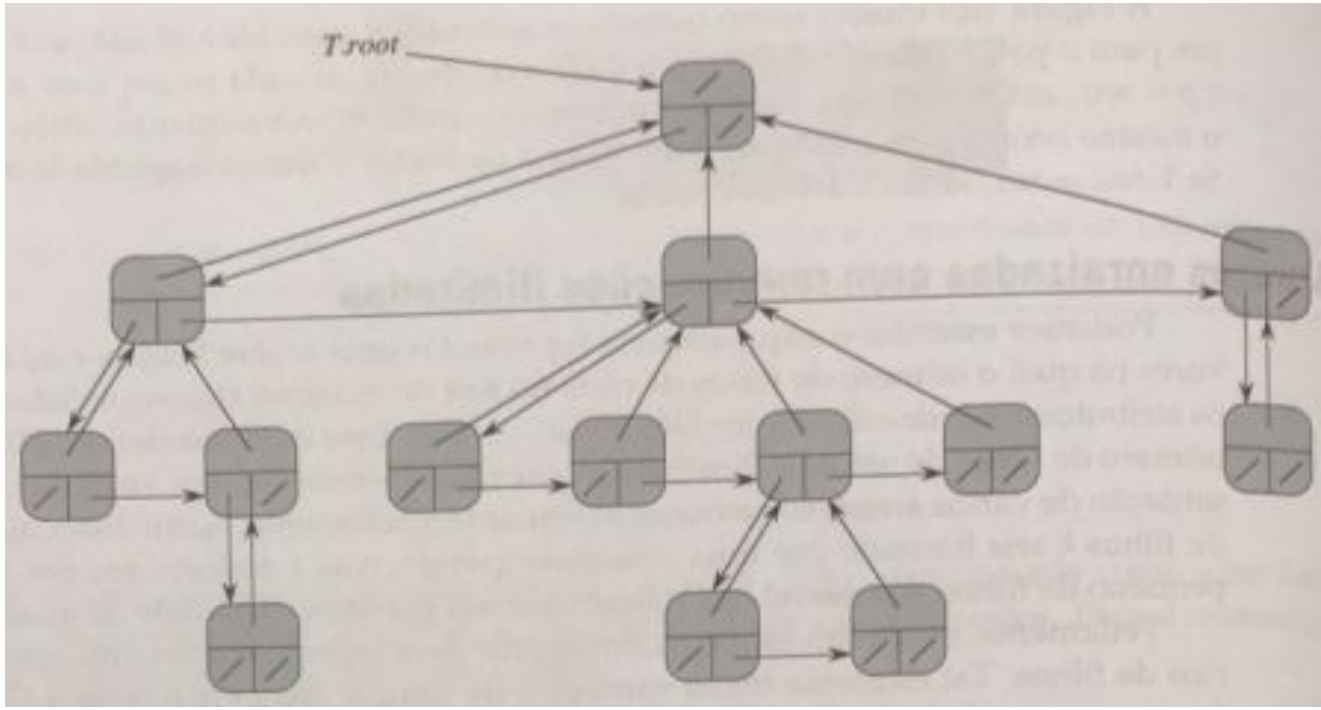



- Raiz  Nó inicial da árvore;
- Grau  número de filhos que um nó possui;
- Nível (profundidade)  distância de um nó até a raiz;
- Altura  maior nível encontrado na árvore;
- Folha  nó que não possui filho;
- Floresta  Conjunto de zero ou mais árvores;
- Caminho  sequência de nós distintos e consecutivos entre os quais existem a relação "é filho de" e/ou "é pai de";
- Árvore cheia  árvore na qual todos os nós têm número máximo de filhos, exceto as folhas, e todas as folhas estão na mesma altura;

# ÁRVORES BINÁRIAS




Árvore binária




Árvore enraizada com ramificações ilimitadas

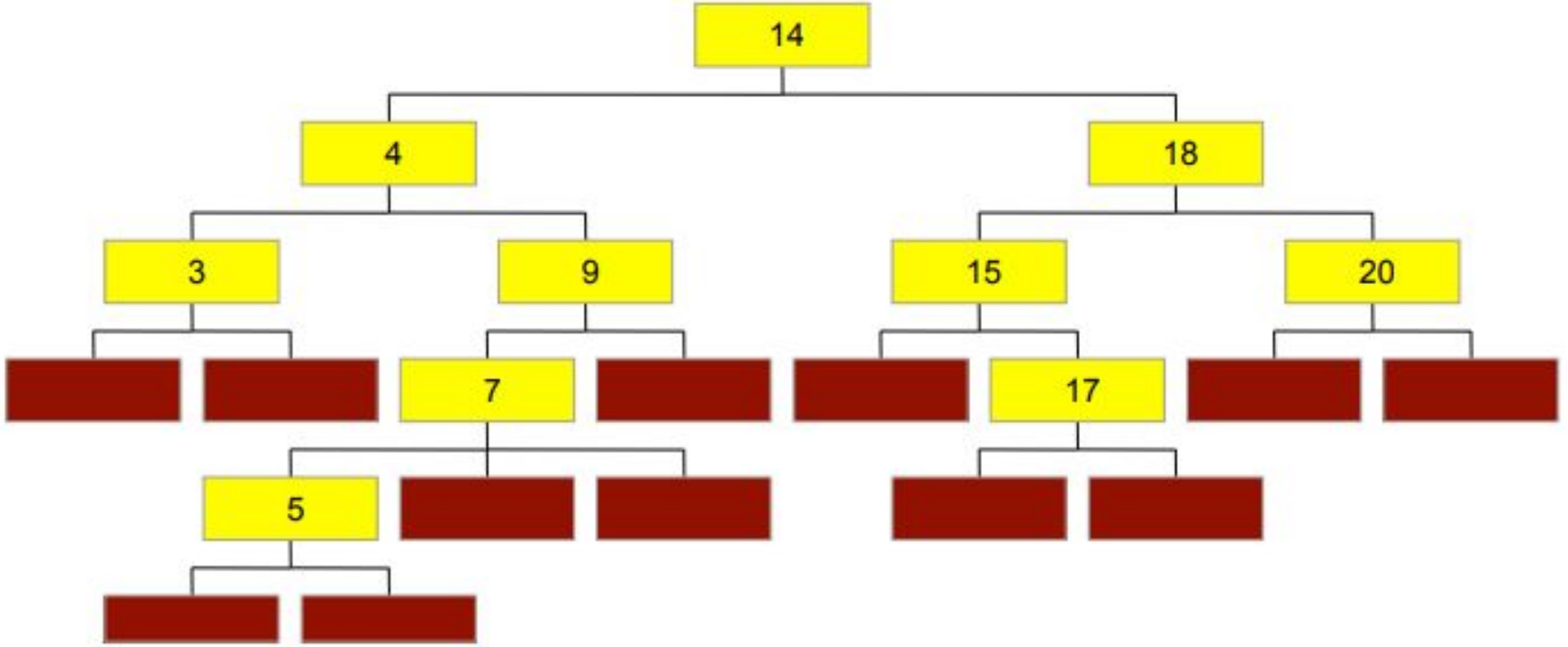
# ÁRVORES BINÁRIAS

Constituídas por três subconjuntos:

- A raiz;
- A sub-árvore esquerda;
- A sub-árvore direita;
- As árvores binárias trazem os valores do nó pai, para o nó filho esquerdo e nó filho direito. Para cada um desses valores, o valor Nil indica que o nó é raiz, que não tem filho do lado esquerdo, ou que não tem filho do lado direito, respectivamente.

# ÁRVORES BINÁRIAS

14, 18, 4, 9, 7, 15, 3, 5, 17, 4, 20, 9, 5



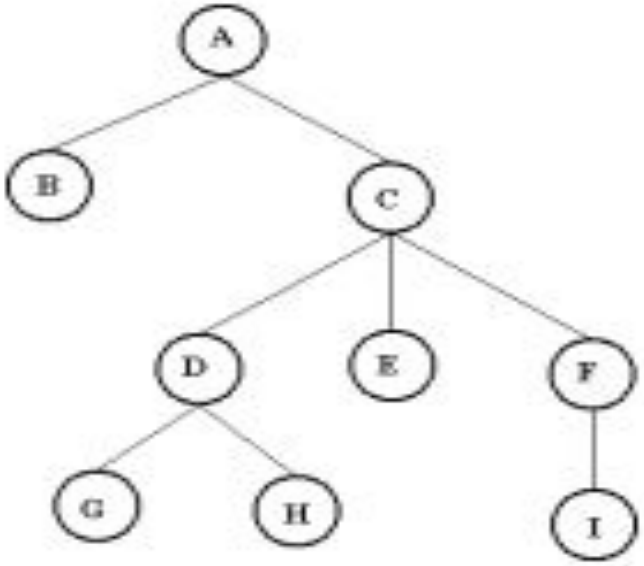
# ÁRVORES BINÁRIAS

Formas de representação

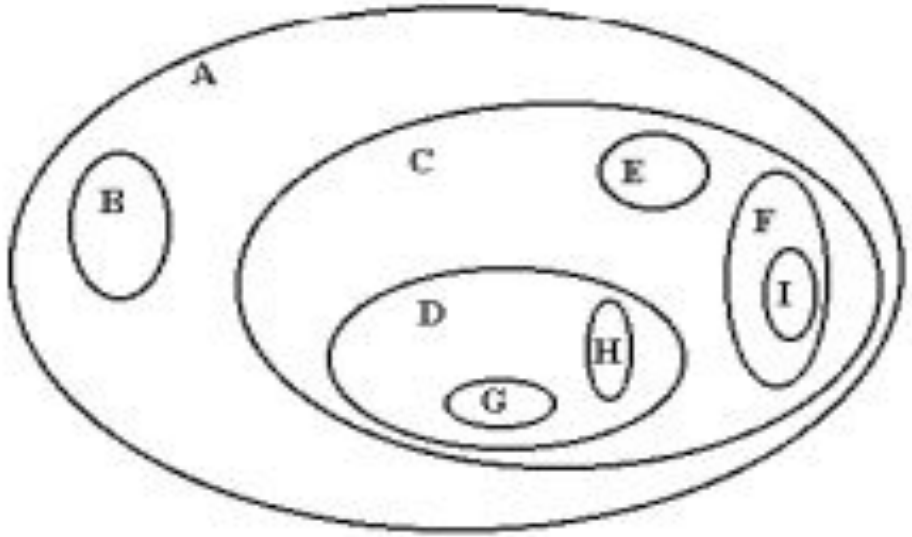
- Por parênteses aninhados

(( A (B) ( C (D (G) (H)) (E) (F (I))))

- Representação hierárquica




- Diagrama de inclusão



# ÁRVORES BINÁRIAS

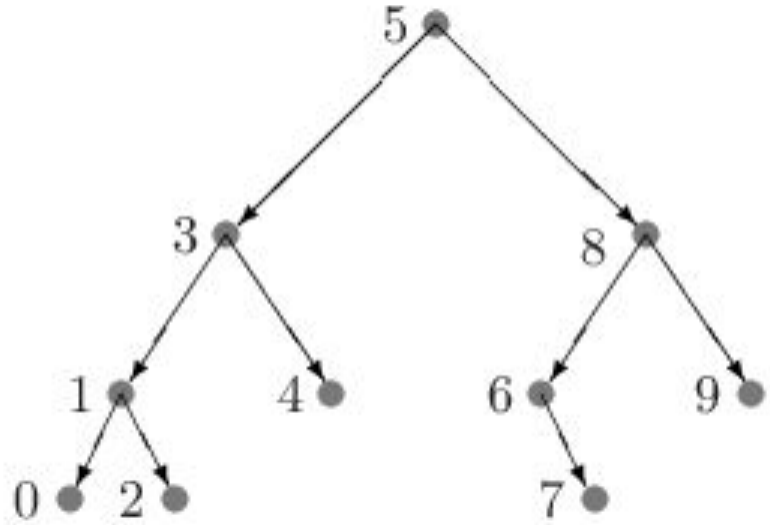
Árvores binárias de busca

Nesse tipo de árvore:

- os valores à esquerda são menores que o valor do seu nó pai;
- os valores à direita são maiores que o valor do nó pai;
- essa regra se aplica a cada uma das suas subárvores.

# ÁRVORES BINÁRIAS

Árvores binárias de busca




Podem ser percorridas de diferentes maneiras:

- esquerda-raiz-direita, também conhecida como:
  - e-r-d;
  - inorder traversal;
  - varredura infixa;
  - varredura central.

```
// Recebe a raiz r de uma árvore binária e
// imprime os conteúdos dos seus nós
// em ordem e-r-d.

void erd (arvore r) {
   if (r != NULL) {
      erd (r->esq);
      printf ("%d\n", r->conteudo);
      erd (r->dir);
   }
}
```

# ÁRVORES BINÁRIAS

Operações:

- Inserção de um nó na árvore;
- Remoção de um nó da árvore;
- Consulta de nós da árvore em ordem;
- Consulta em pré-ordem;
- Consulta em pós-ordem;

# ÁRVORES BINÁRIAS

**Pré-ordem**:

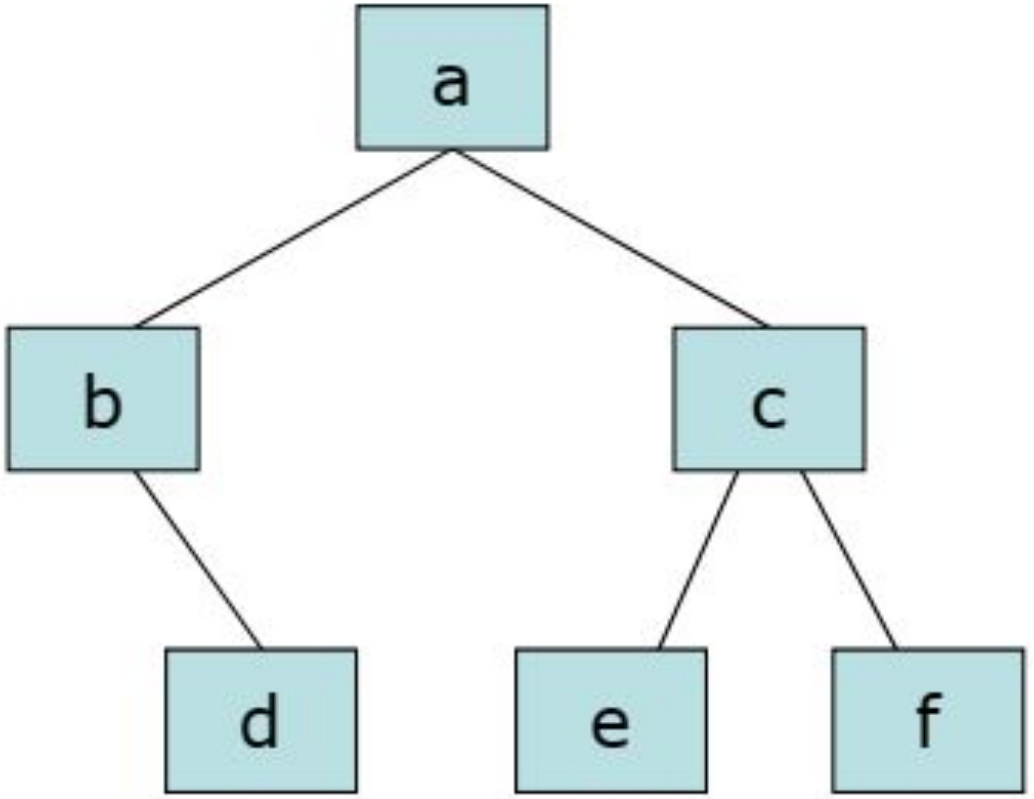
- trata raiz, percorre sae, percorre sad;
- exemplo: a b d c e f;

**Ordem simétrica (ou In-Ordem):**

- percorre sae, trata raiz, percorre sad;
- exemplo: b d a e c f ;

**Pós-ordem:**

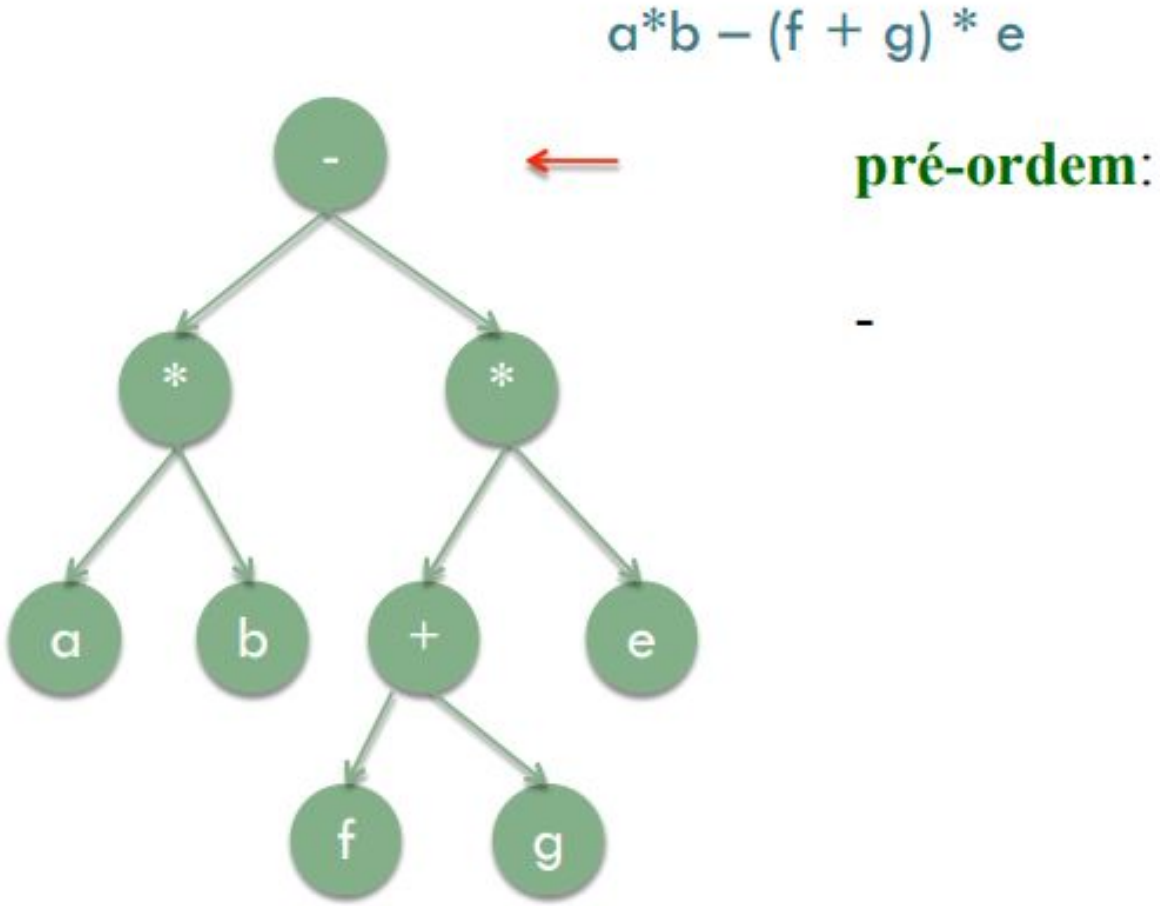
- – percorre sae, percorre sad, trata raiz;
  – exemplo: d b e f c a;



# ÁRVORES BINÁRIAS

**Pré-ordem:**

```
pre_ordem (pt)
{
 if (pt == NULL) return ();
visite(pt);
pre_ordem (pt->esq);
pre_ordem (pt-> dir);
}
```

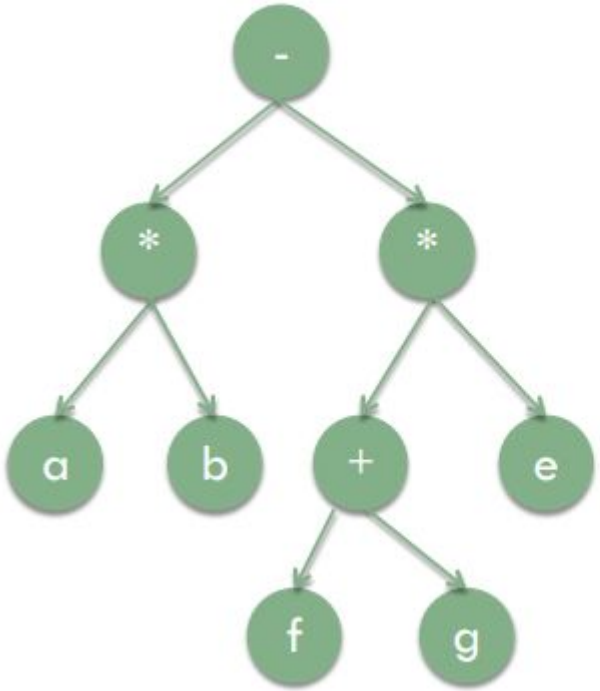


# ÁRVORES BINÁRIAS

**Em-ordem:**

```
em_ordem (pt)
{
 if (pt == NULL) return ();
em_ordem (pt->esq);
visite(pt);
em_ordem (pt-> dir);
}
```

a*b – (f + g) * e




**em-ordem:**

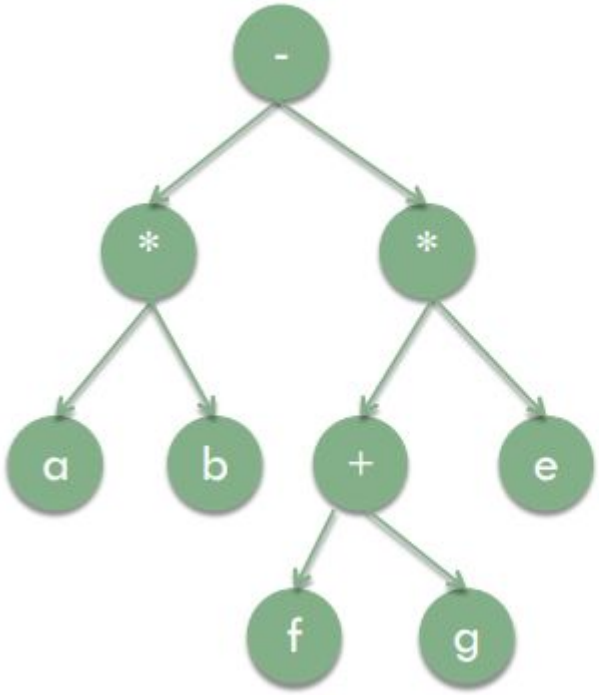
-a*b - f+g * e

# ÁRVORES BINÁRIAS

**Pós-ordem**:

```
pos_ordem (pt)
{
 if (pt == NULL) return ();
pos_ordem (pt->esq);
pos_ordem (pt-> dir);
visite(pt);
}
```

a*b – (f + g) * e



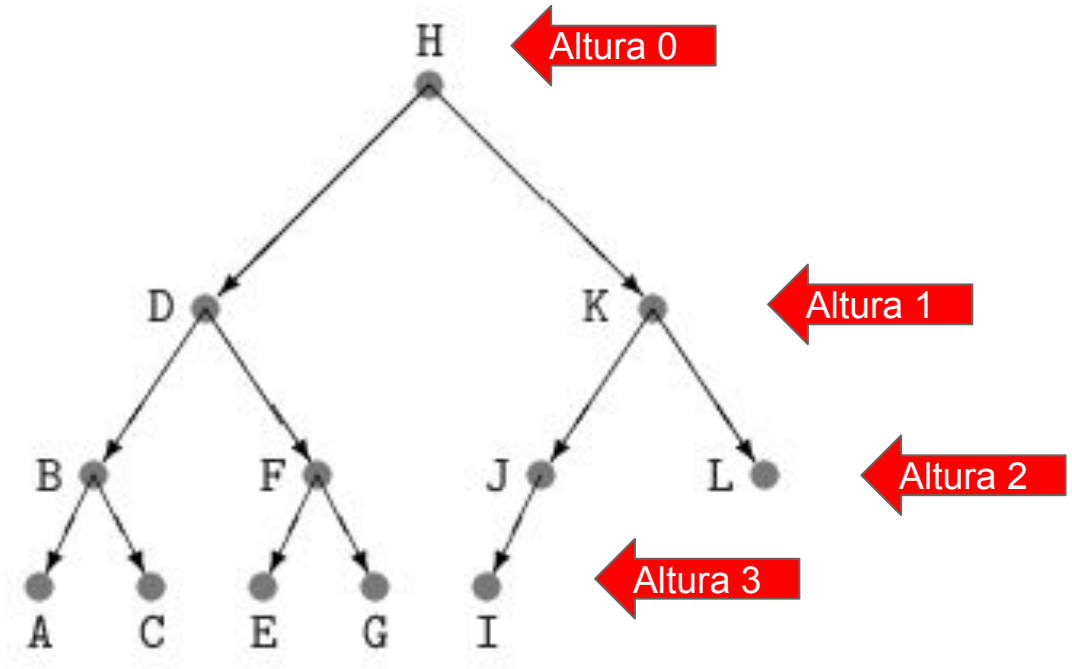

**pós-ordem**:

a b * f g + e * -

# ÁRVORES BINÁRIAS

**Altura da árvore**:

- Distância entre o nó raiz da árvore/sub-árvore e o seu nó descendente mais distante;




**n−1 ≥ h ≥ lg (n)**

**onde:**
**n - 1 => nó sem filhos.**

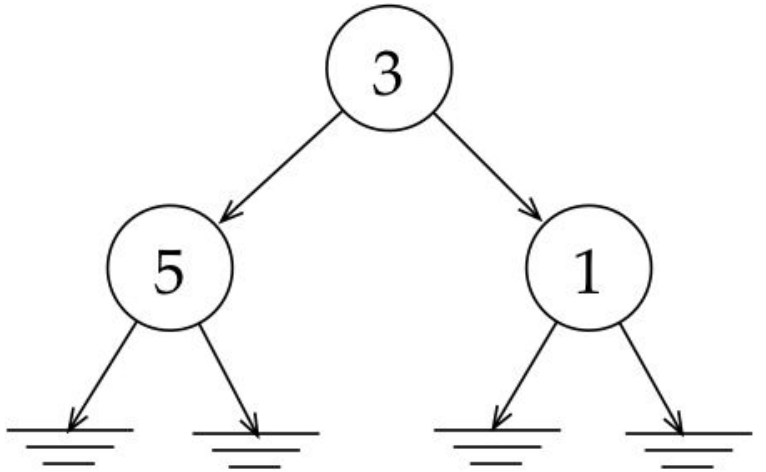
# ÁRVORES BINÁRIAS

**Exemplo de código:**

```
class NodoArvore:
    def __init__(self, chave=None, esquerda=None, direita=None):
        self.chave = chave
        self.esquerda = esquerda
        self.direita = direita

    def __repr__(self):
        return '%s <- %s -> %s' % (self.esquerda and self.esquerda.chave,
                                   self.chave,
                                   self.direita and self.direita.chave)
```

```
raiz = NodoArvore(3)
raiz.esquerda = NodoArvore(5)
raiz.direita  = NodoArvore(1)
print("Árvore: ", raiz)
```



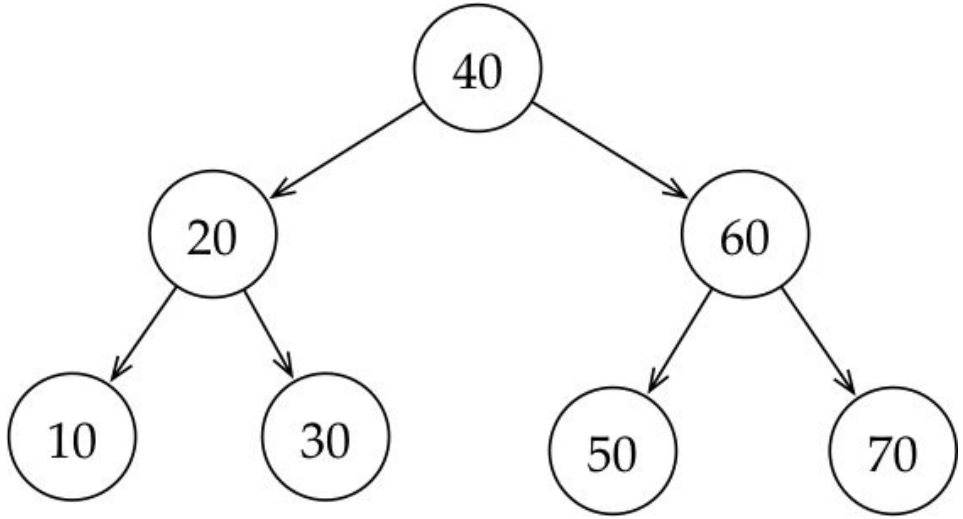
# ÁRVORES BINÁRIAS

**Exemplo de código:**

```
raiz = NodoArvore(40)

raiz.esquerda = NodoArvore(20)
raiz.direita  = NodoArvore(60)

raiz.direita.esquerda  = NodoArvore(50)
raiz.direita.direita   = NodoArvore(70)
raiz.esquerda.esquerda = NodoArvore(10)
raiz.esquerda.direita  = NodoArvore(30)
```



# ÁRVORES BINÁRIAS

**Exemplo de código:**

```
def em_ordem(raiz):
    if not raiz:
        return

    # Visita filho da esquerda.
    em_ordem(raiz.esquerda)

    # Visita nodo corrente.
    print(raiz.chave),

    # Visita filho da direita.
    em_ordem(raiz.direita)
```

```
10
20
30
40
50
60
70
```

# ÁRVORES BINÁRIAS

**Inserindo dados**:

```
def insere(raiz, nodo):
    """Insere um nodo em uma árvore binária de pesquisa."""
    # Nodo deve ser inserido na raiz.
    if raiz is None:
        raiz = nodo

    # Nodo deve ser inserido na subárvore direita.
    elif raiz.chave < nodo.chave:
        if raiz.direita is None:
            raiz.direita = nodo
        else:
            insere(raiz.direita, nodo)

    # Nodo deve ser inserido na subárvore esquerda.
    else:
        if raiz.esquerda is None:
            raiz.esquerda = nodo
        else:
            insere(raiz.esquerda, nodo)
```

# ÁRVORES BINÁRIAS

**Criando a árvore:**

```
# Cria uma árvore binária de pesquisa.
raiz = NodoArvore(40)
for chave in [20, 60, 50, 70, 10, 30]:
    nodo = NodoArvore(chave)
    insere(raiz, nodo)
# Imprime o caminhamento em ordem da árvore.
em_ordem(raiz)
```

# ÁRVORES BINÁRIAS

**Buscando um valor na árvore:**

```
def busca(raiz, chave):
    """Procura por uma chave em uma árvore binária de
pesquisa."""
    # Trata o caso em que a chave procurada não está
presente.
    if raiz is None:
        return None

    # A chave procurada está na raiz da árvore.
    if raiz.chave == chave:
        return raiz

    # A chave procurada é maior que a da raiz.
    if raiz.chave < chave:
        return busca(raiz.direita, chave)

    # A chave procurada é menor que a da raiz.
    return busca(raiz.esquerda, chave)
```

# ÁRVORES BINÁRIAS

**Buscando um valor na árvore:**

```
# Cria uma árvore binária de pesquisa.
raiz = NodoArvore(40)
for chave in [20, 60, 50, 70, 10, 30]:
    nodo = NodoArvore(chave)
    insere(raiz, nodo)

# Procura por valores na árvore.
for chave in [-50, 10, 30, 70, 100]:
    resultado = busca(raiz, chave)
    if resultado:
        print("Busca pela chave {}: Sucesso!".format(chave))
    else:
        print("Busca pela chave {}: Falha!".format(chave))
```

```
Busca pela chave -50: Falha!
Busca pela chave 10: Sucesso!
Busca pela chave 30: Sucesso!
Busca pela chave 70: Sucesso!
Busca pela chave 100: Falha!
```

# Referências


- https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html
- https://www.google.com/search?q=%C3%A1rvores+bin%C3%A1rias&oq=%C3%A1rvores+bin%C3%A1rias&aqs=chrome..69i57j0l3j0i22i30l6.4863j0j15&sourceid=chrome&ie=UTF-8
- https://www.inf.ufsc.br/~aldo.vw/estruturas/Arvores/Estruturas.ArvBin.html
- http://www.ic.uff.br/~boeres/slides_ed/ed_ArvoresPercursos.pdf
- https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node32.html
- https://algoritmosempython.com.br/cursos/algoritmos-python/estruturas-dados/arvores/