# 5-Árvores


CCA0916 - ESTRUTURA DE DADOS I
BCC

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br

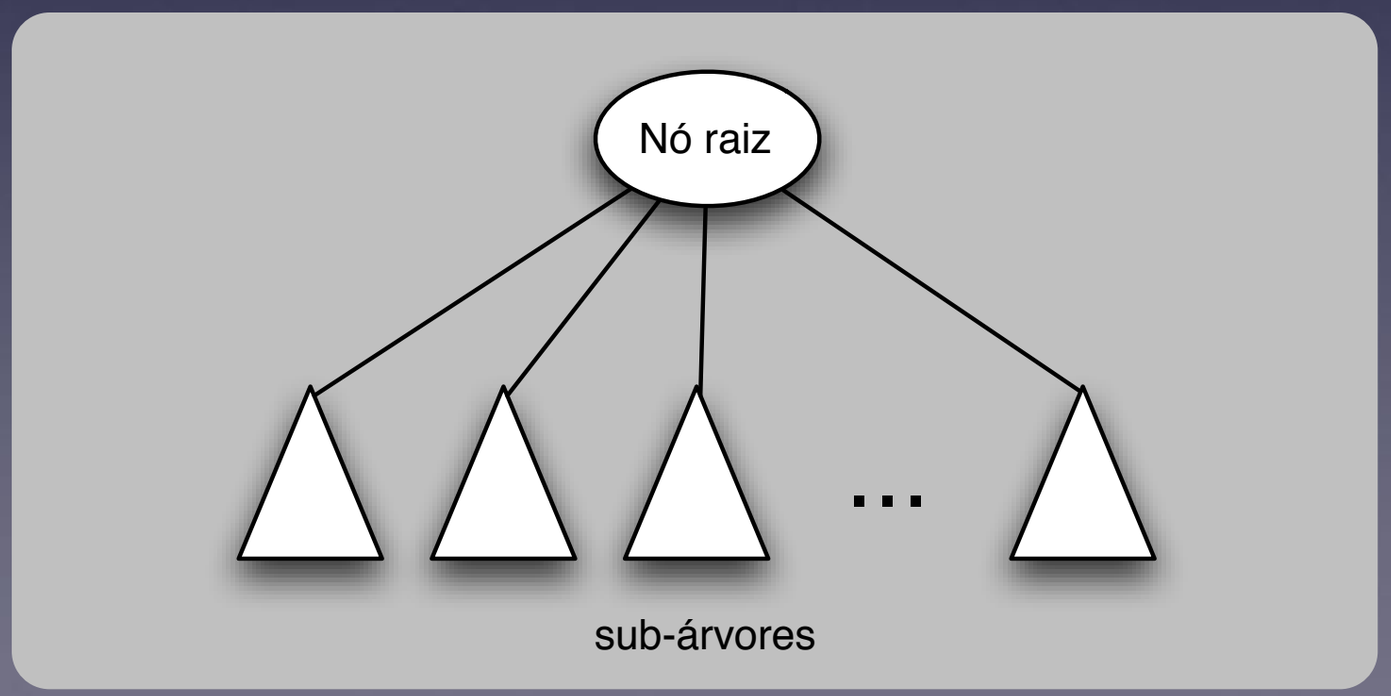
Blumenau, 2023

# Tópicos

- Introdução
- Árvores binárias
  - representaçao
  - ordens de percurso
  - altura de uma árvore
- Árvores com número variável de filhos
  - representação
  - altura da árvore

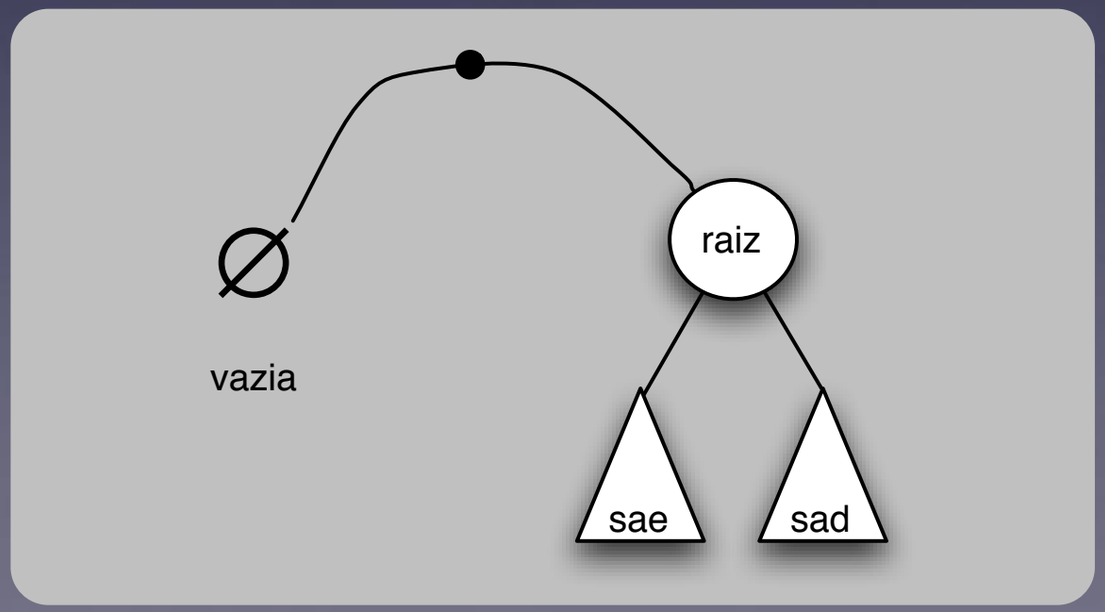
# Introdução - Árvore

- Árvore é um conjunto de nós tal que
  - existe um nó r, denominado **raiz** com zero ou mais sub-árvores, cujas raízes estão ligadas a r
  - os nós raízes destas sub-árvores são os ***filhos*** de r
  - os ***nós internos*** da árvore são os nós com filhos
  - as ***folhas*** ou *nós externos* são os nós sem filhos



# Árvores binárias

- Cada nó tem zero, um ou dois filhos
- Definição: uma árvore binária é...
  - uma árvore vazia, ou
  - um nó raiz com duas sub-árvores
    - a sub-árvore da direita (sad)
    - a sub-árvore da esquerda (sae)

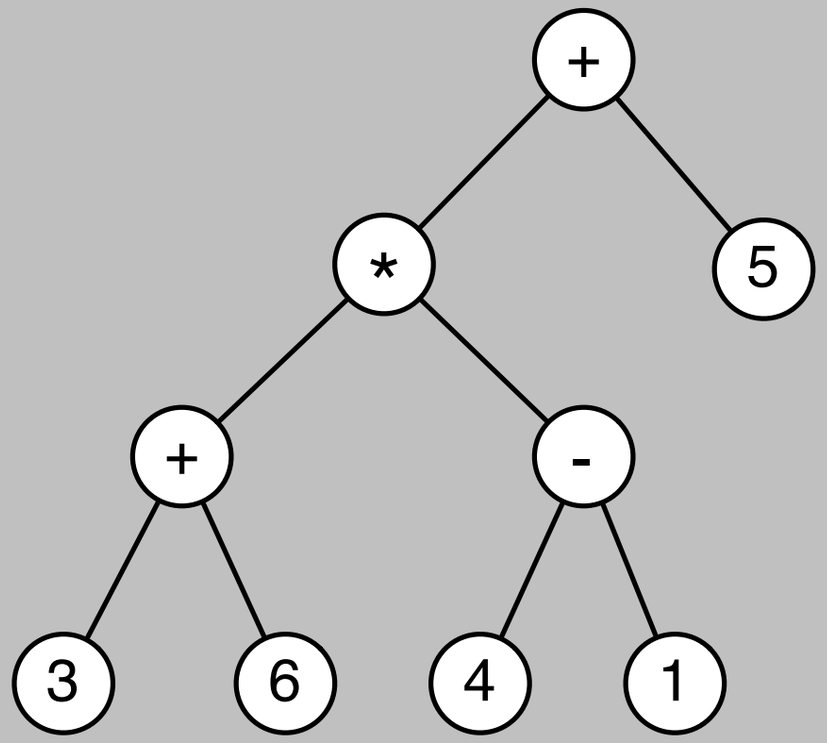


# Árvores binárias

- Exemplo: expressões aritmáticas
  - nós folhas representam operandos
  - nós internos operadores

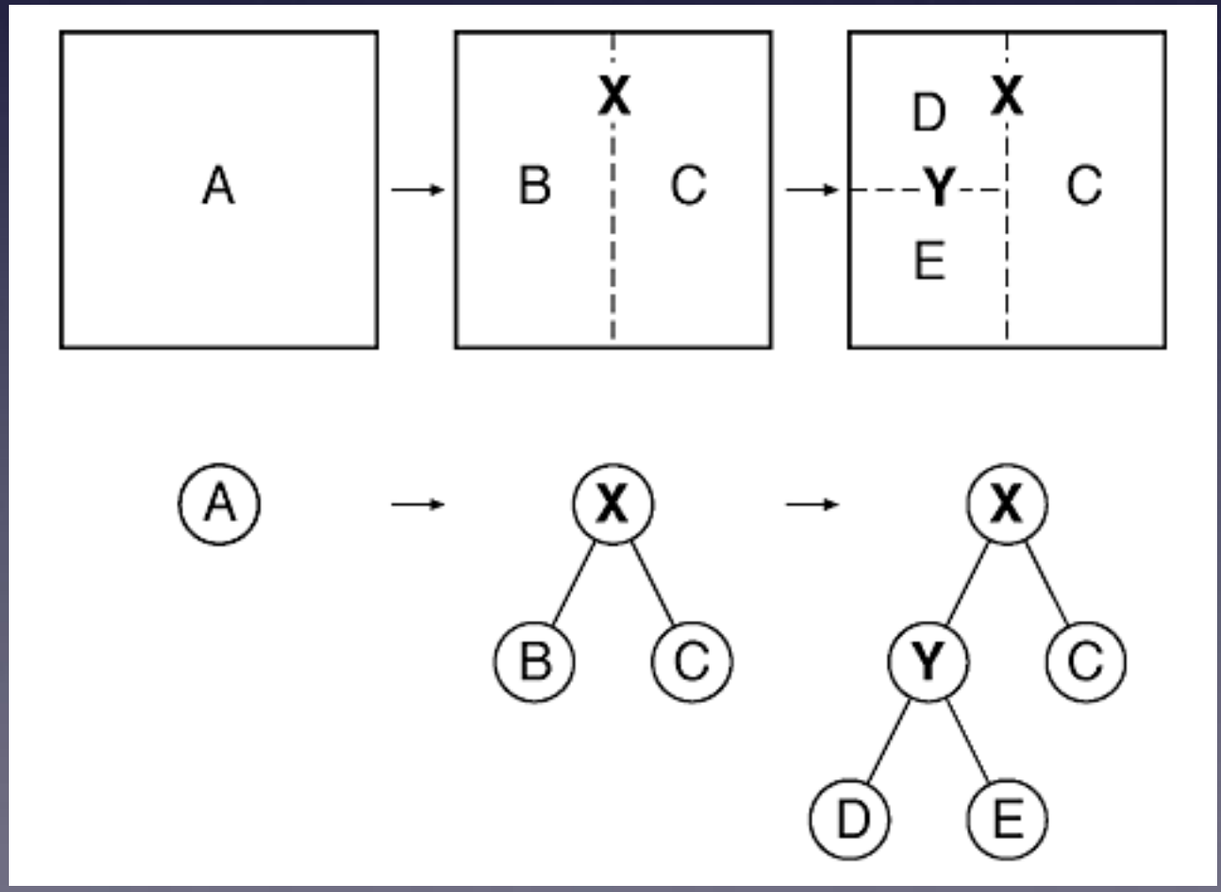
- exemplo:

  (3+6)*(4-1)+5



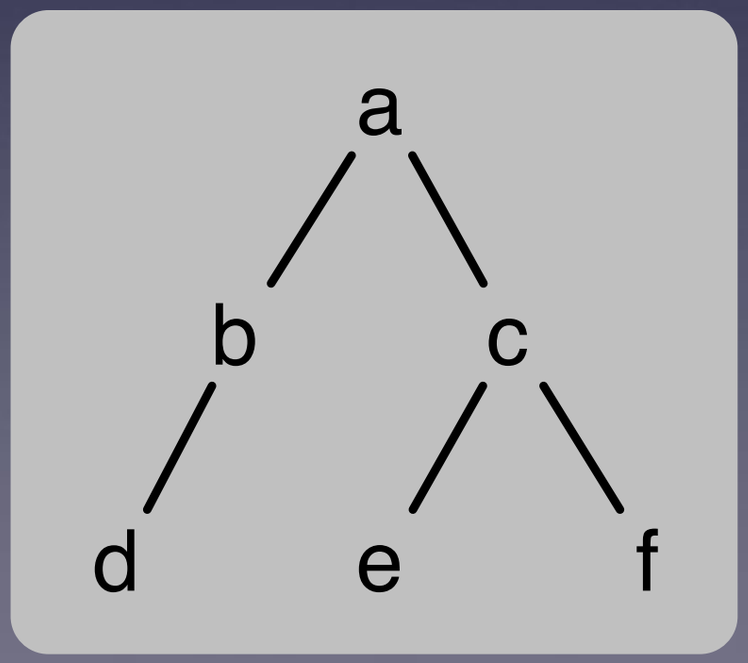
# Árvores binárias

Árvore BSP (Binary Space Partition Tree): é uma estrutura de dados que representa uma sub-divisão recursiva e hierárquica de um espaço n-dimensional em sub-espaços convexos.



# Árvores binárias

- Notação textual
  - árvore vazia representada por < >
  - árvores não vazias por <raiz sae sad>
- exemplo:
  <a <b <d<><>> <> > <c <e<><>><f<><>>>>



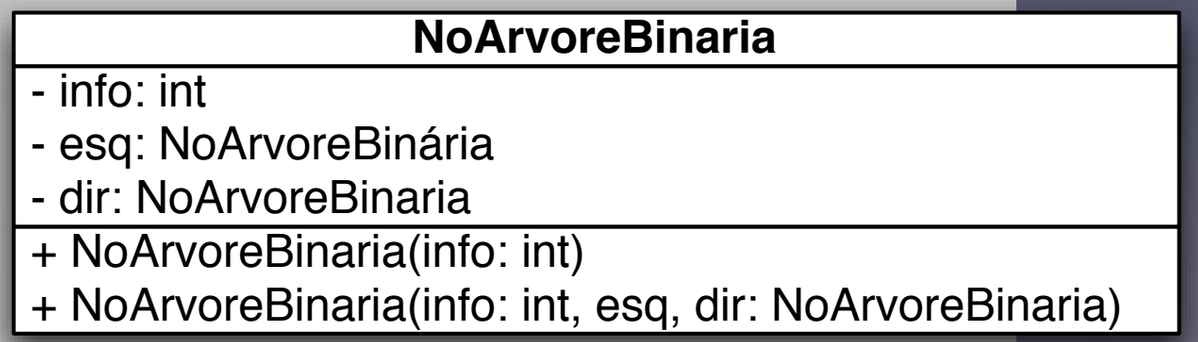
# Implementação

- Representação da árvore: ponteiro para o nó raiz
- Representação de um nó da árvore:

## Classe NoArvoreBinaria

| **NoArvoreBinaria** |
| --- |
| - info: int<br>- esq: NoArvoreBinária<br>- dir: NoArvoreBinaria |
| + NoArvoreBinaria(info: int)<br>+ NoArvoreBinaria(info: int, esq, dir: NoArvoreBinaria) |

```java
public class NoArvoreBinaria {
    private int info;
    private NoArvoreBinaria esq;
    private NoArvoreBinaria dir;

public NoArvoreBinaria(int info) {
    this.info = info;
    esq = null;
    dir = null;
}

public NoArvoreBinaria(int info,
                       NoArvoreBinaria esq,
                       NoArvoreBinaria dir) {
        this.info = info;
        this.esq = esq;
        this.dir = dir;
    }
```

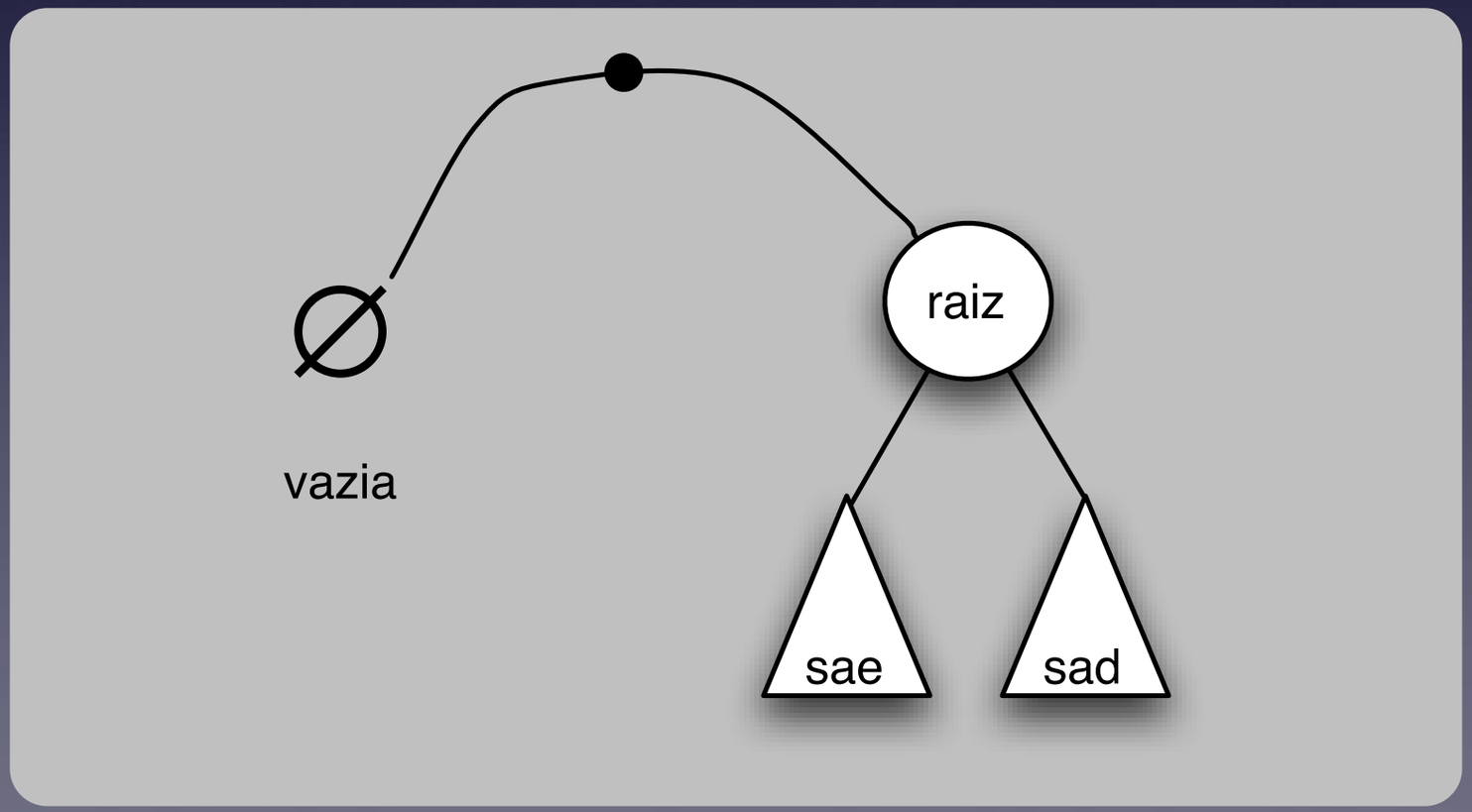


# Classe ÁrvoreBinária

| **ArvoreBinaria** |
| --- |
| - raiz: NoArvoreBinaria |
| + ArvoreBinaria()<br>+ insere(v : int):NoArvoreBinaria<br>+ insere(v : int, esq, dir: NoArvoreBinaria): NoArvoreBinaria<br>+ vazia() : boolean<br>+ toString(): String<br>- imprimePre(no: NoArvoreBinaria) : String<br>+ pertence(info : int) : boolean<br>- pertence(no : NoArvoreBinaria, info: int) : boolean |

# Implementação

- Geralmente a implementação é recursiva
- Utiliza a definição recursiva da estrutura



# Construtor da classe

- cria uma árvore vazia

```
public ArvoreBinaria() {
    raiz = null;
  }
```

# Método Insere

- cria um nó dadas a informação e as duas sub-árvores
- retorna o endereço do nó criado
- sempre atribui o novo nó para a raiz

```
public NoArvoreBinaria insere(int v,
                              NoArvoreBinaria esq,
                              NoArvoreBinaria dir) {
    NoArvoreBinaria no = new NoArvoreBinaria(v, esq, dir);
    raiz = no;
    return no;
}
```

# Método vazia

- Indica se uma árvore está vazia ou não

```
public boolean vazia () {
     return (raiz == null);
}
```

# Método pertence

- método para verificar a existência de uma valor *info* em um dos nós
- utiliza um método privado auxiliar recursivo

**Algoritmo:** public boolean pertence(int info)

**retorna** *pertence(raiz, info)*;

**Algoritmo 4.1**: Pertence (público)

**Algoritmo:** private boolean pertence(NoArvoreBinaria no, int info)

**se** $(no == null)$ **então**
  **retorna** *falso*;
**senão**
  **retorna** $((no.info == info)$ **ou** $pertence(no.esq, info)$**ou**
  $pertence(no.dir, info))$;

**Algoritmo 4.2**: Pertence (privado)

# Método toString

**Algoritmo:** public String toString()

**retorna** *imprimePre(raiz)*;

**Algoritmo 4.3**: Método toString (público)

**Algoritmo:** private String imprimePre(NoArvoreBinaria no)

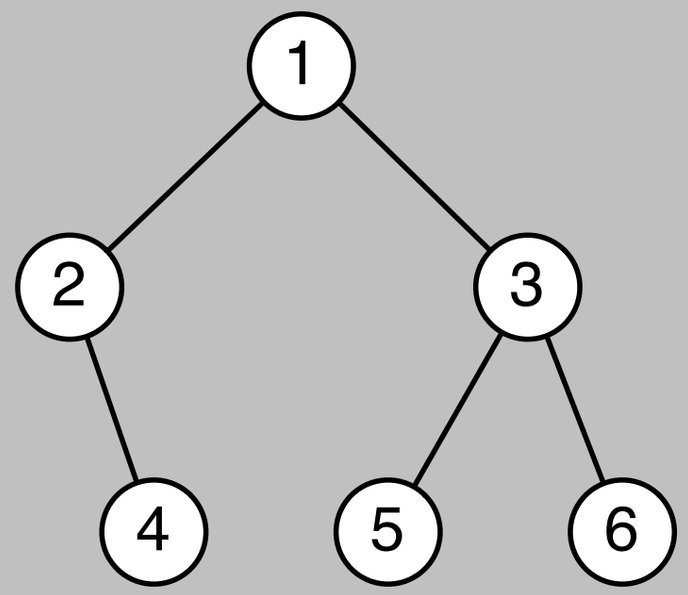
*String s = new String(“”);*
$s \leftarrow s +$ “ < ”;
**se** $(no \neq null)$ **então**
    $s \leftarrow s + no.info$;
    $s \leftarrow s + imprimePre(no.esq)$;
    $s \leftarrow s + imprimePre(no.dir)$;
$s \leftarrow s +$ “ > ”;
**retorna** $s$;

**Algoritmo 4.4**: Método imprimePre (privado)

# Exemplo

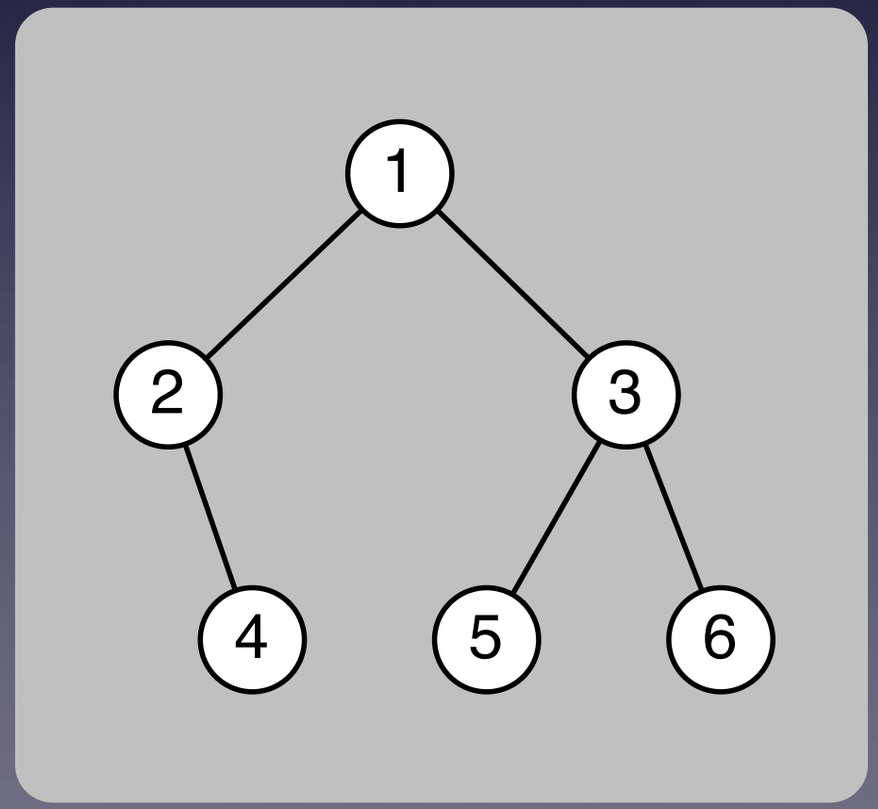
1<2<><4<><><>>><3<5<><><>><6<><><>>>>

```
ArvoreBinaria a = new ArvoreBinaria();

a.insere(1,
          a.insere(2,
                    null,
                    a.insere(4,null, null)
                   ),
          a.insere(3,
                    a.insere(5, null, null),
                    a.insere(6, null, null)
                    )
         );
```



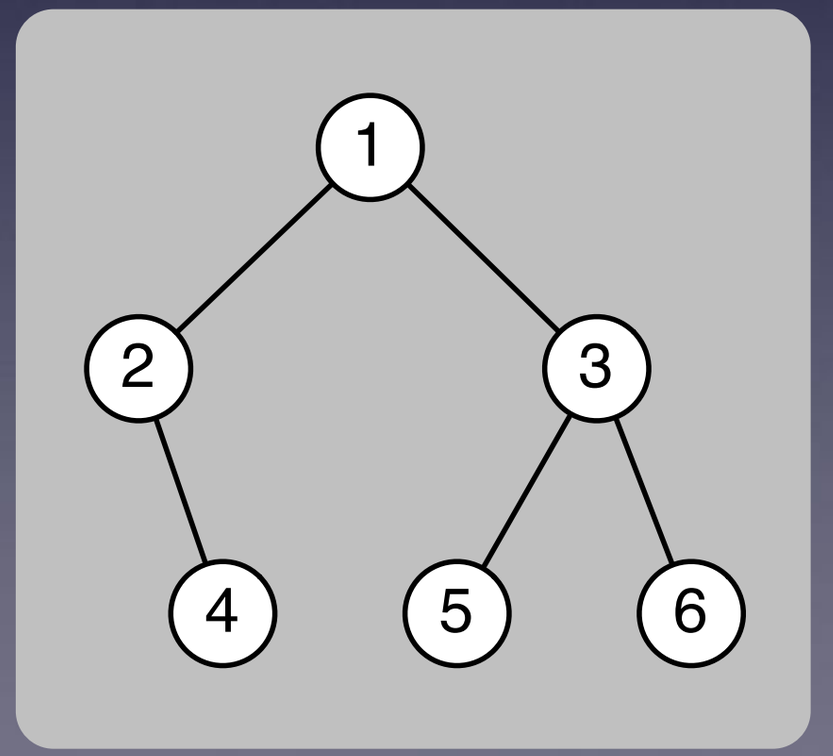
# Ordem de percurso

- Pré-ordem: trata raiz, percorre sae, percorre sad
  exemplo: 1 2 4 3 5 6
- Ordem simétrica: percorre sae, trata raiz, percorre sad
  exemplo 2 4 1 5 3 6
- Pós-ordem: percorre sae, percorre sad, trata raiz
  exemplo: 4 2 5 6 3 1



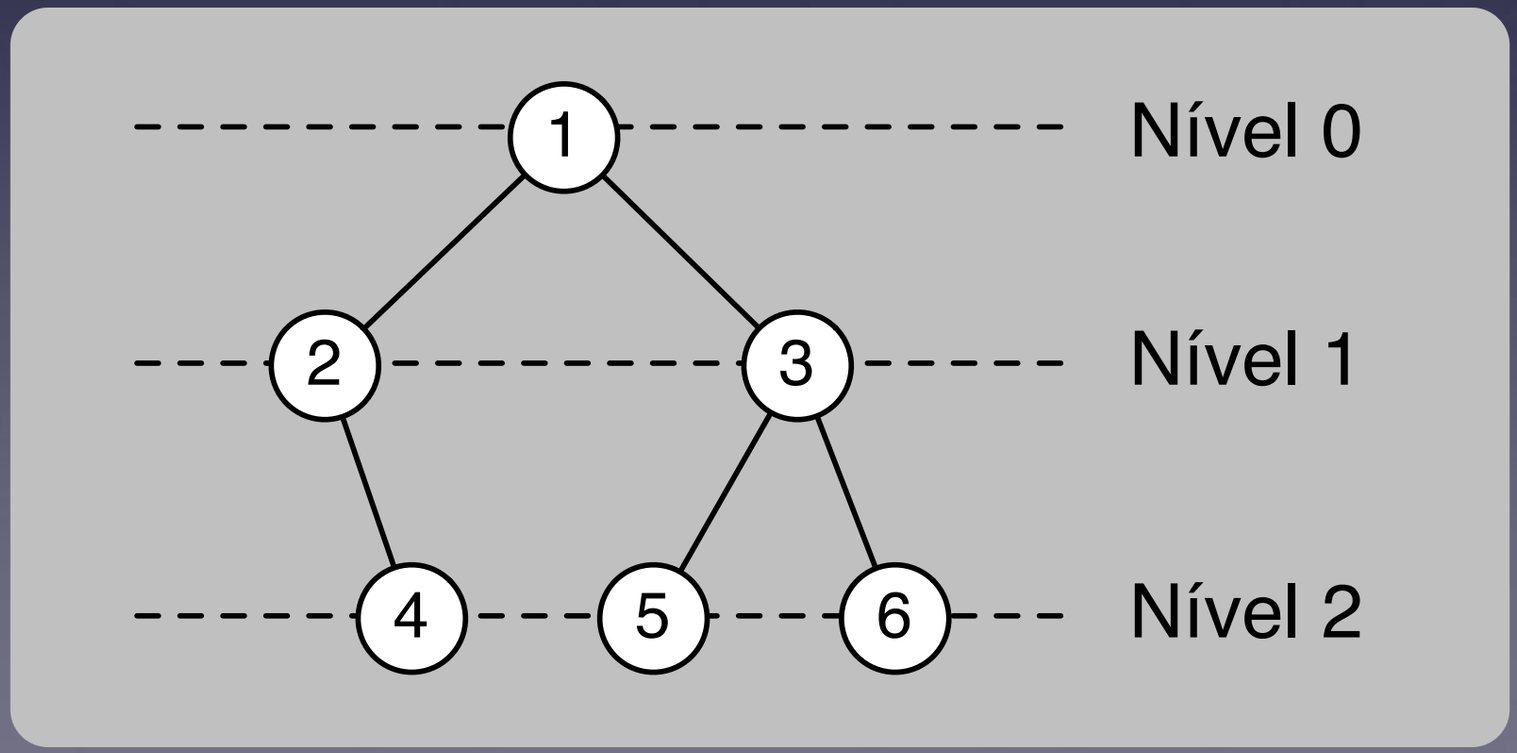
# Altura

- propriedade fundamental das árvores: só existe um caminho da raiz para qualquer nó
- Altura: comprimento do caminho mais longo da raiz até uma das folhas
- a altura de uma árvore com um único nó raiz é zero
- a altura de uma árvore vazia é -1
- exemplo: h = 2



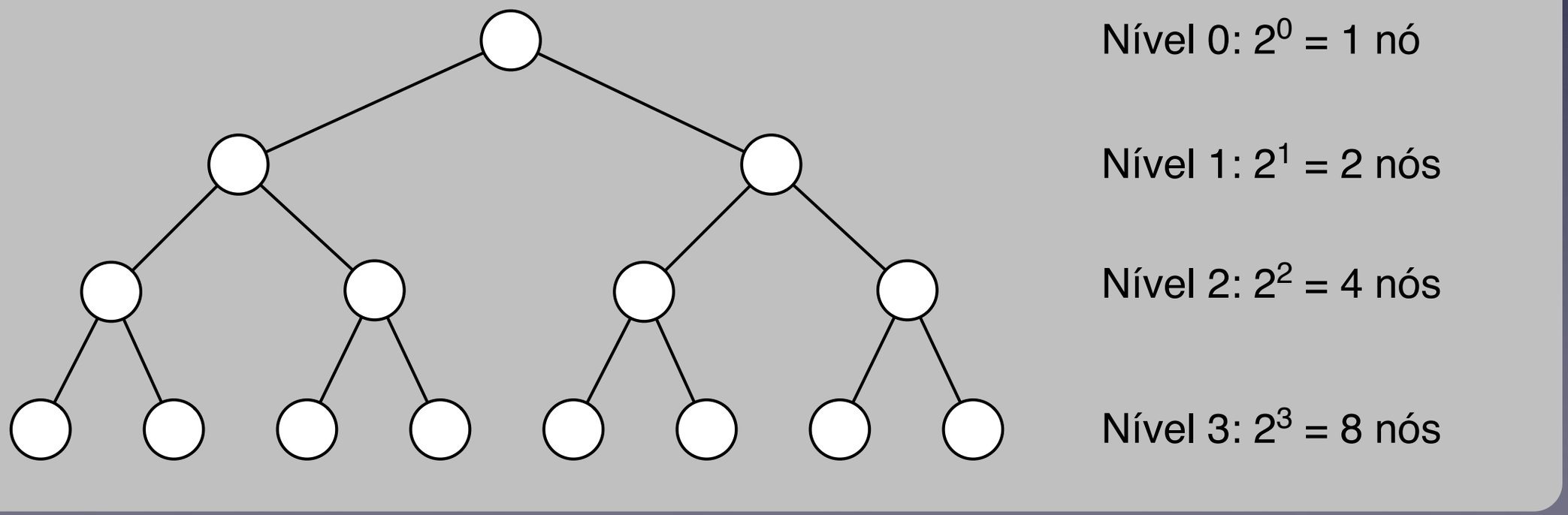
# Altura - Nível

- Nível de um nó:
  - a raiz está no nível 0, seus filhos estão no nível 1, ...
  - o último nível da árvore é a altura da árvore



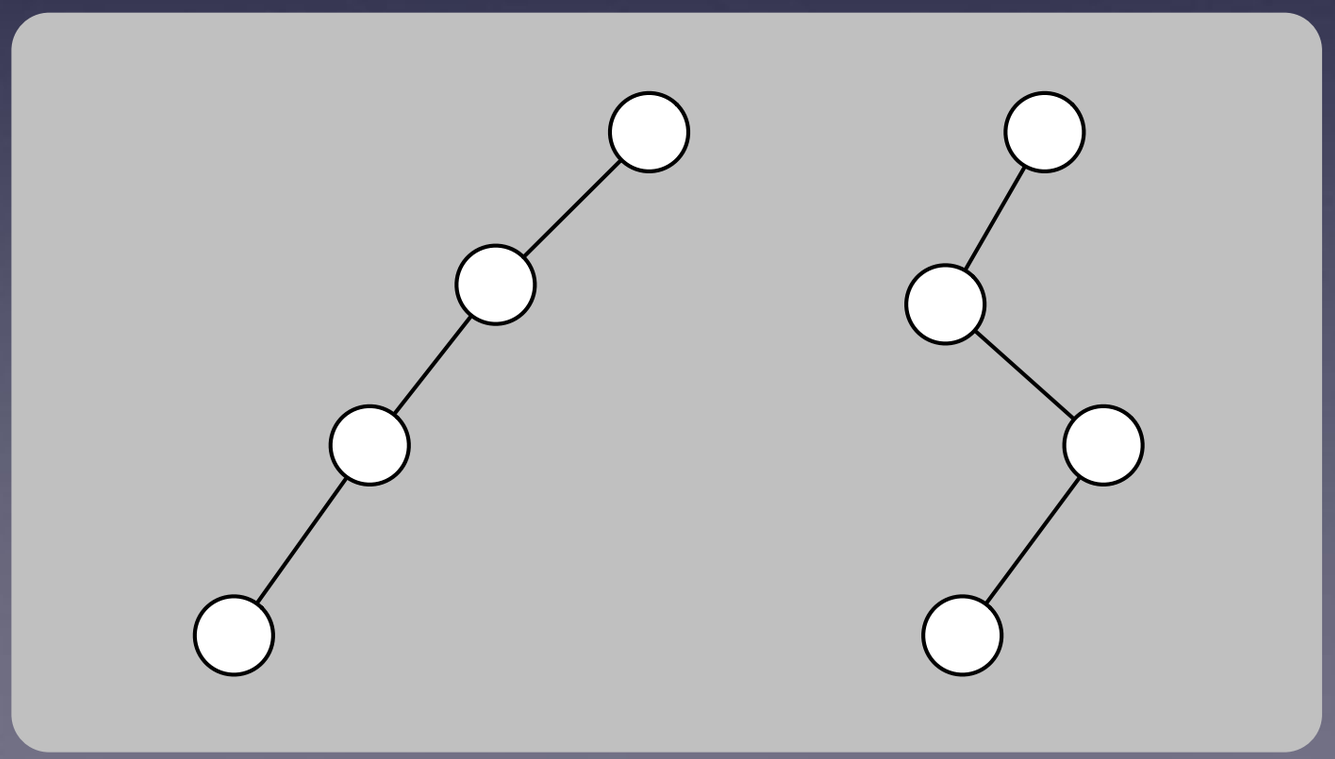
# Balanceamento

- Árvore cheia
- Todos os seus nós internos têm duas sub-árvores associadas
- o número *n* de nós de uma árvore binária cheia de altura *h* é $n = 2^{h+1} - 1$



# Balanceamento

- Árvore degenerada
- Todos os seus nós internos têm uma única sub-árvore associada
- o número *n* de nós de uma árvore binária degenerada de altura *h* é $n = h + 1$

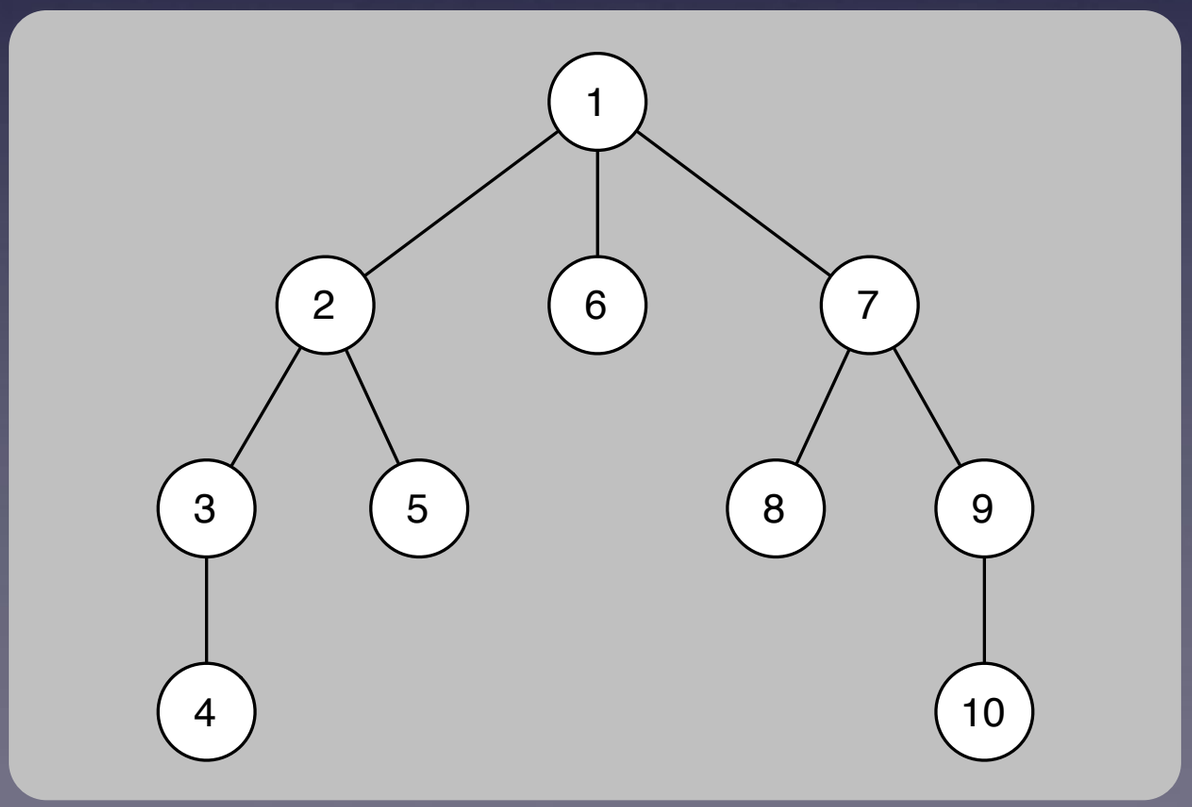


# Balanceamento

- Esforço computacional necessário para alcançar qualquer nó da árvore é proporcional à altura da árvore
- Altura da árvore binária com n nós:
  - mínima: proporcional a *log n* (árvore cheia)
  - máxima: proporcional a *n* (árvore degenerada)

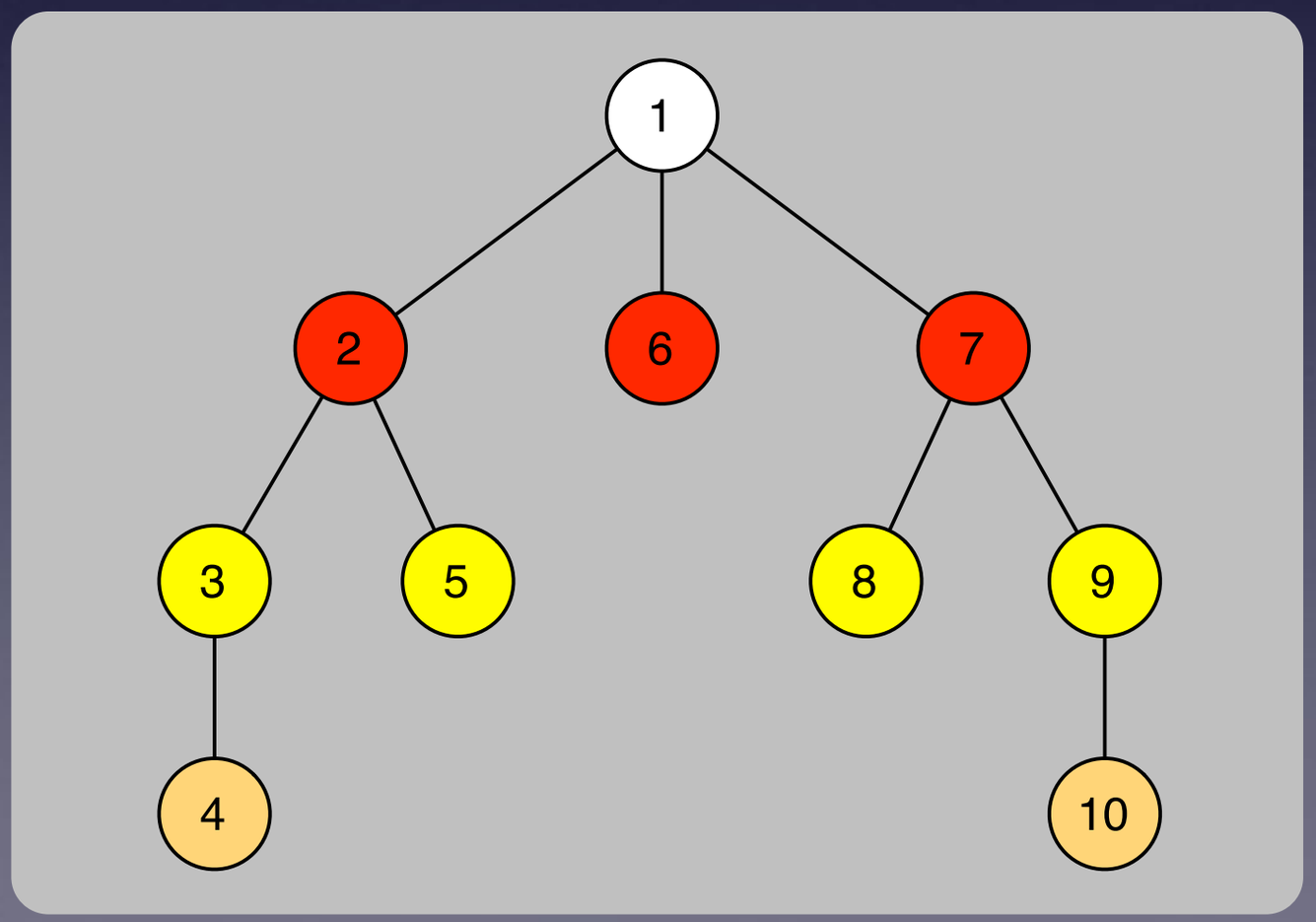
# Árvores com número variável de filhos

- cada nó pode ter mais do que duas sub-árvores associadas
- sub-árvores de um nós são dispostas em ordem: primeira sub-árvore (sa1), segunda sub-árvore (sa2), etc...

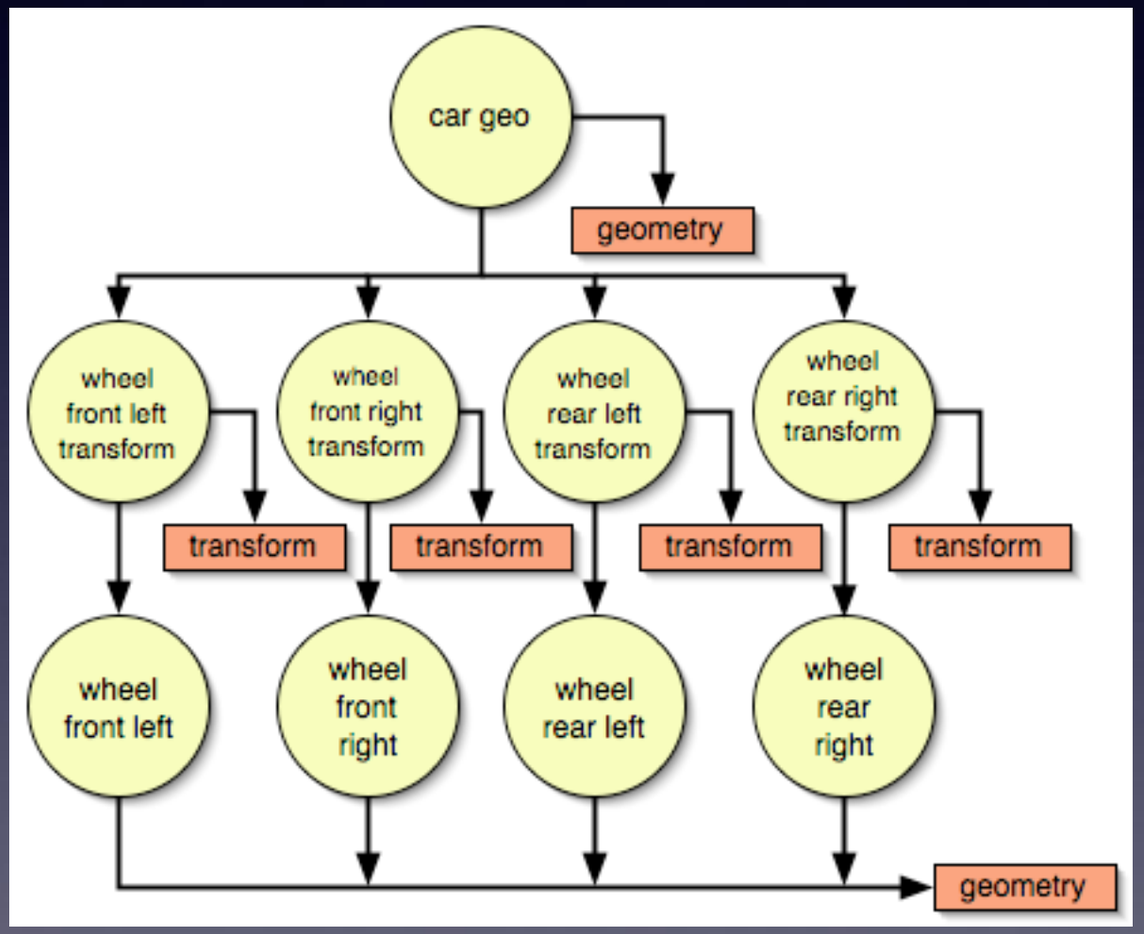


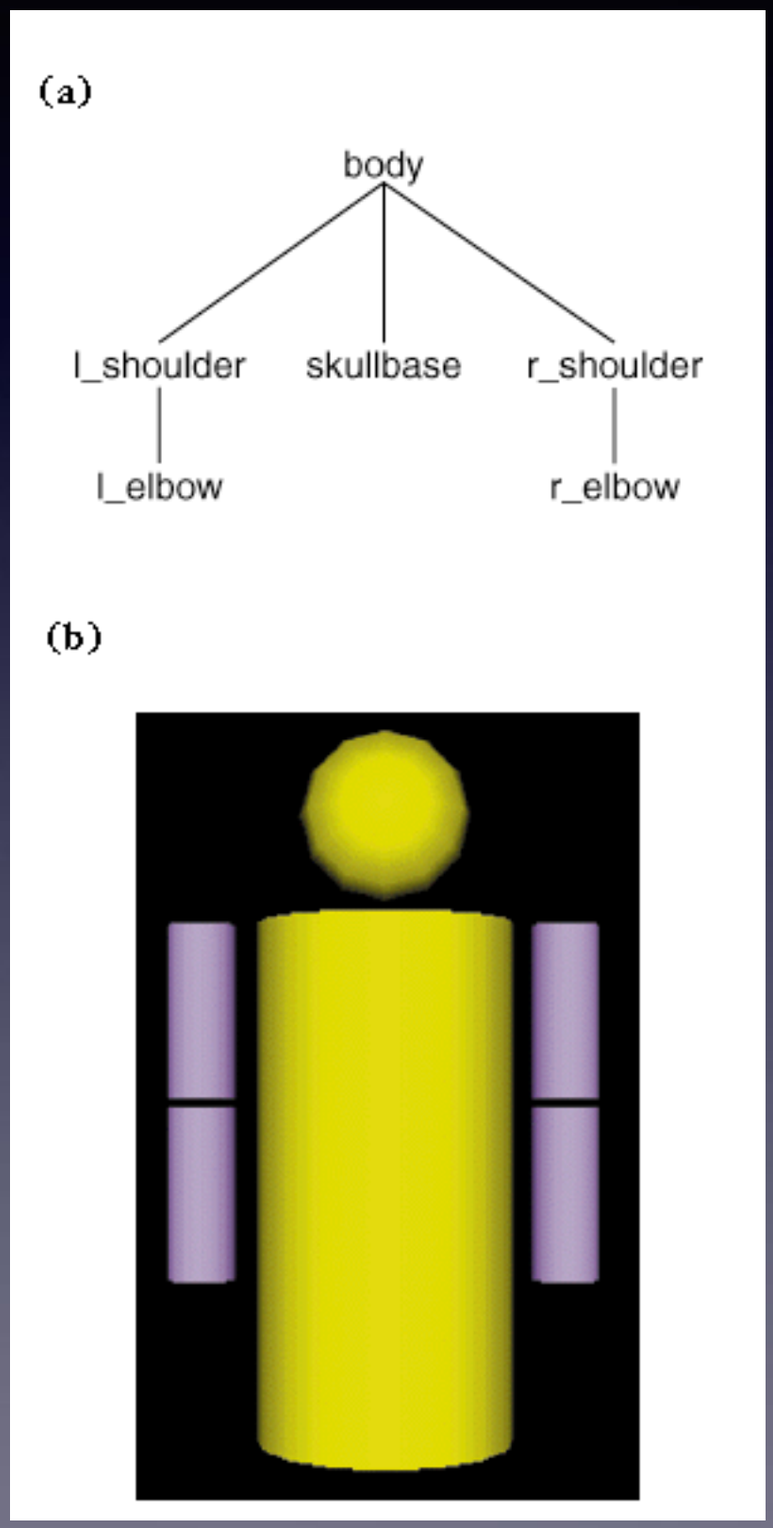
# Árvores com número variável de filhos

- Notação textual: `<raiz sa1 sa2 ... san>`
- Exemplo:
  α = <1 <2 <3 <4> > <5> > <6> <7 <8> <9 <10> > > >



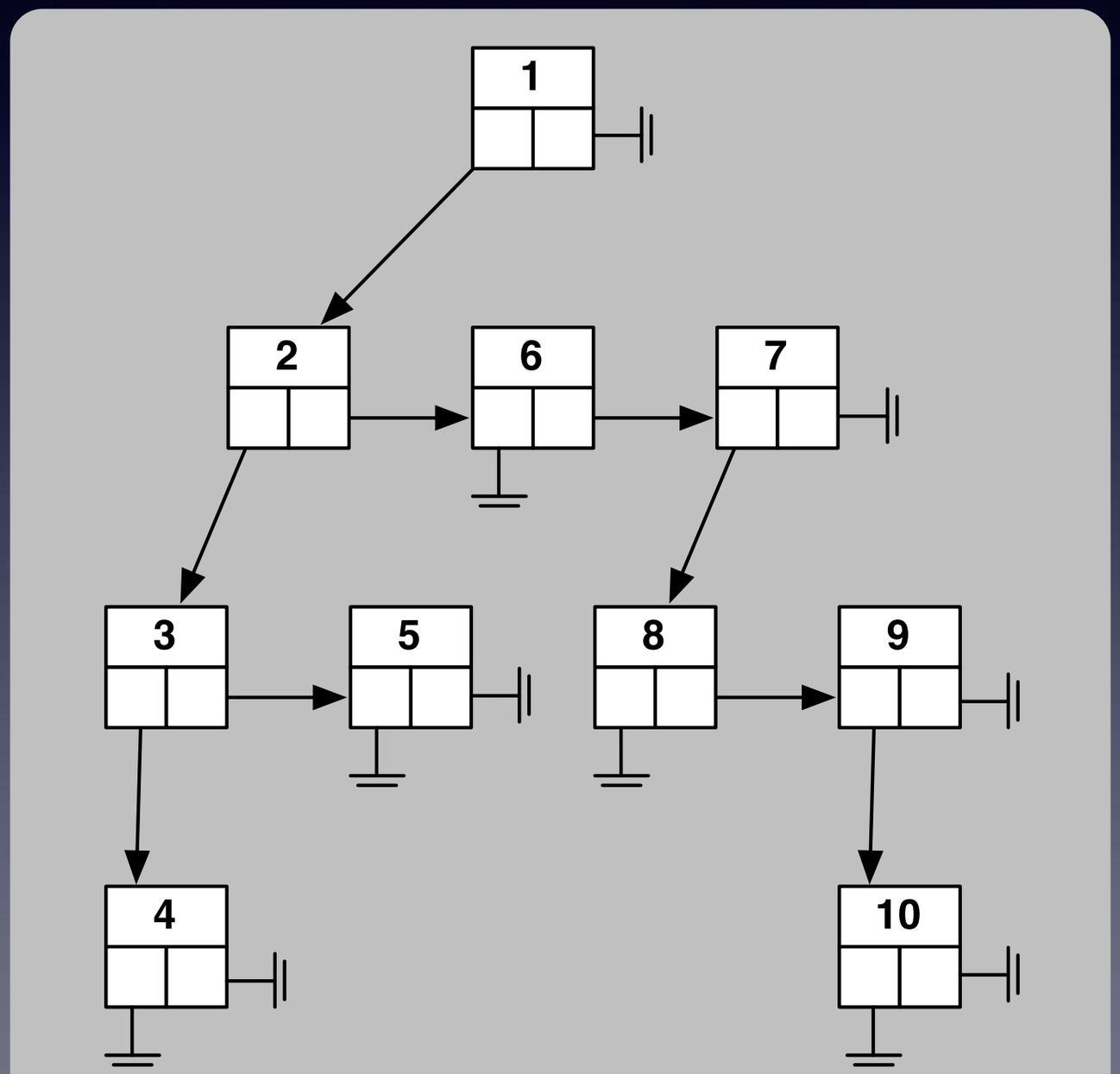
# Exemplo: grafo de cena



# Representação (modelagem)

- Representação de árvore com número variável de filhos: utiliza uma "lista de filhos"




- um nó aponta apenas para seu primeiro filho (prim)
- cada um dos filhos aponta para o próximo irmao (prox)

# Implementação

- Representação de um nó da árvore: Classe NoArvore
- ponteiro para a primeira sub-árvore filha (null de o nó for uma folha)
- ponteiro para a próxima sub-árvore irmã (null se for o último filho)

| **NoArvore** |
| --- |
| - info: int<br>- prim: NoArvore<br>- prox: NoArvore |
| + NoArvore(info: int)<br>+ NoArvore(info: int, sa: NoArvore) |

```java
public class NoArvore {
    private int info;
    private NoArvore prim;
    private NoArvore prox;

    public NoArvore(int info) {
        this.info = info;
        prim = null;
        prox = null;
    }

    public NoArvore(int info, NoArvore sa) {
        this.info = info;
        prox = sa.prim;
        prim = sa;
    }

    // métodos set/get...

}
```

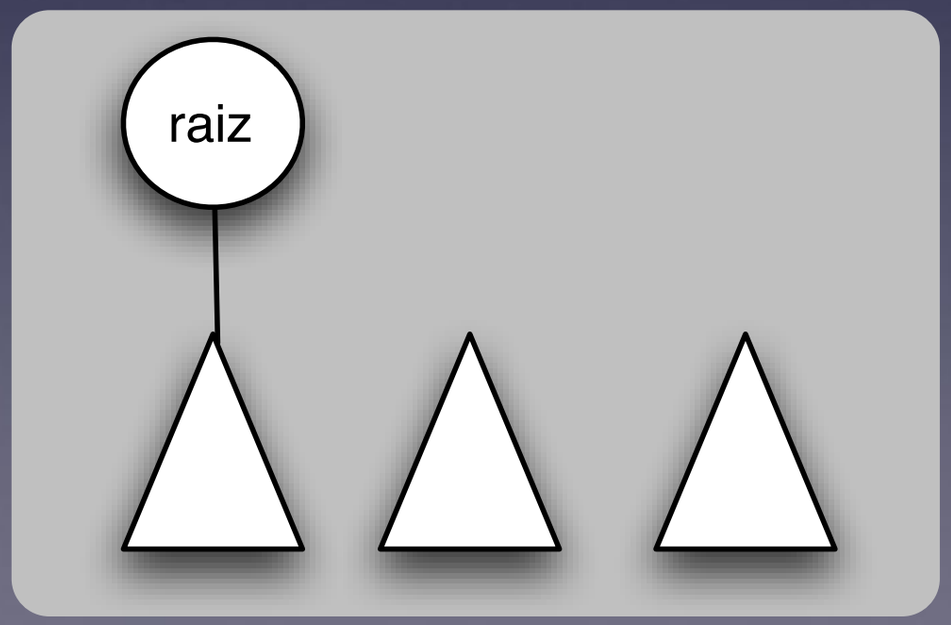
# Classe Árvore

| **Arvore** |
| --- |
| - raiz: NoArvore |
| + Arvore()<br>+ criaNo(v : int): NoArvore<br>+ insereFilho(pai : int, filho: NoArvore): NoArvore<br>+ toString(): String<br>- imprime(no: NoArvore) : String<br>+ pertence(info : int) : boolean<br>- pertence(no : NoArvore, info: int) : boolean<br>+ altura() : int<br>- altura(no : NoArvore) : int |

# Implementação

- Implementação dos métodos geralmente é recursiva
- Usa a definição recursiva da estrutura

- Uma árvore é: um nó raiz, zero ou mais sub-árvores



# Implementação

- uma árvore não pode ser vazia
- uma folha é identificada como um nó com zero sub-árvores (uma folha não é um nó com sub-árvores vazias como nas árvores binárias)
- os métodos não consideram o caso de árvores vazias

# Implementação

- método criaNo
- cria uma folha:
  - chama o construtor da classe NoArvore
  - inicializa os atributos do nó, atribuindo null a prim e prox

```
public NoArvore criaNo(int info) {
        NoArvore novo = new NoArvore(info);
        raiz = novo;
        return novo;
    }
```

# Implementação

- método insereFilho
- insere uma sub-árvore como filha de um nó dado, sempre no início da lista, por simplicidade

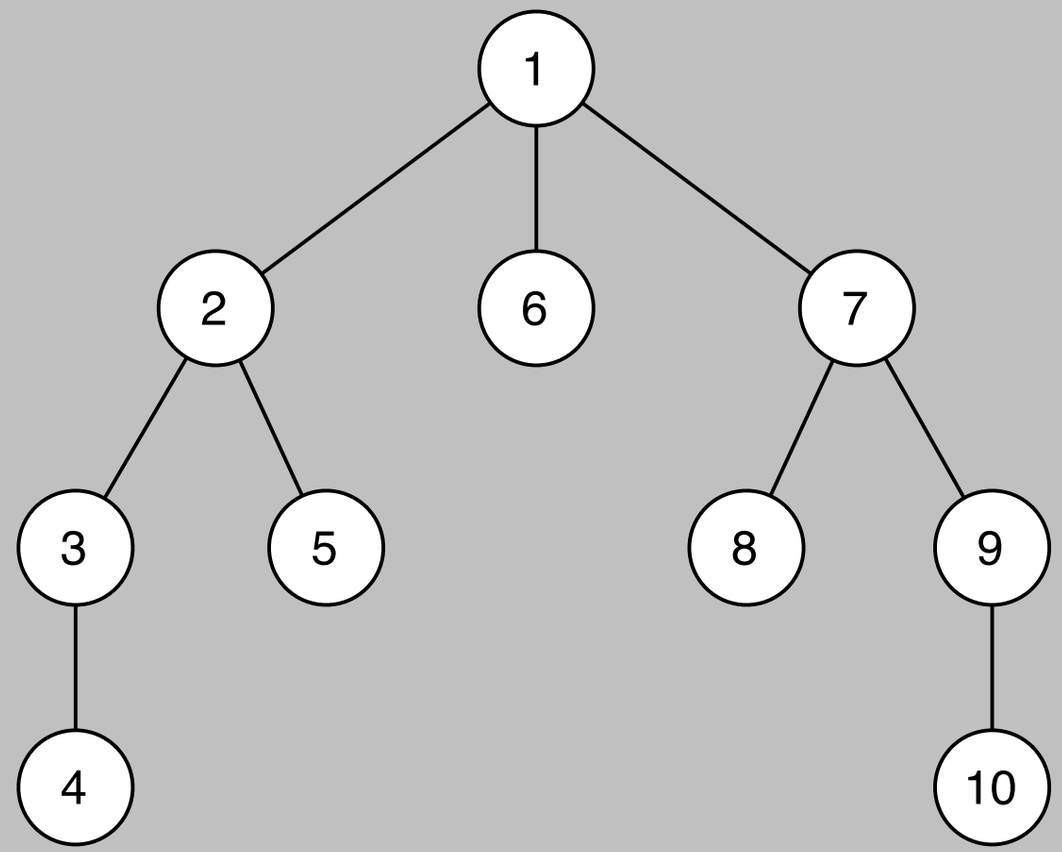
**Algoritmo:** public void insere(NoArvore pai, NoArvore filho)

*String s = new String("");*
$filho.prox \leftarrow pai.prim$;
$pai.prom \leftarrow filho$;
$raiz \leftarrow pai$;

**Algoritmo 4.5**: Árvore: método insere

# Exemplo de criação

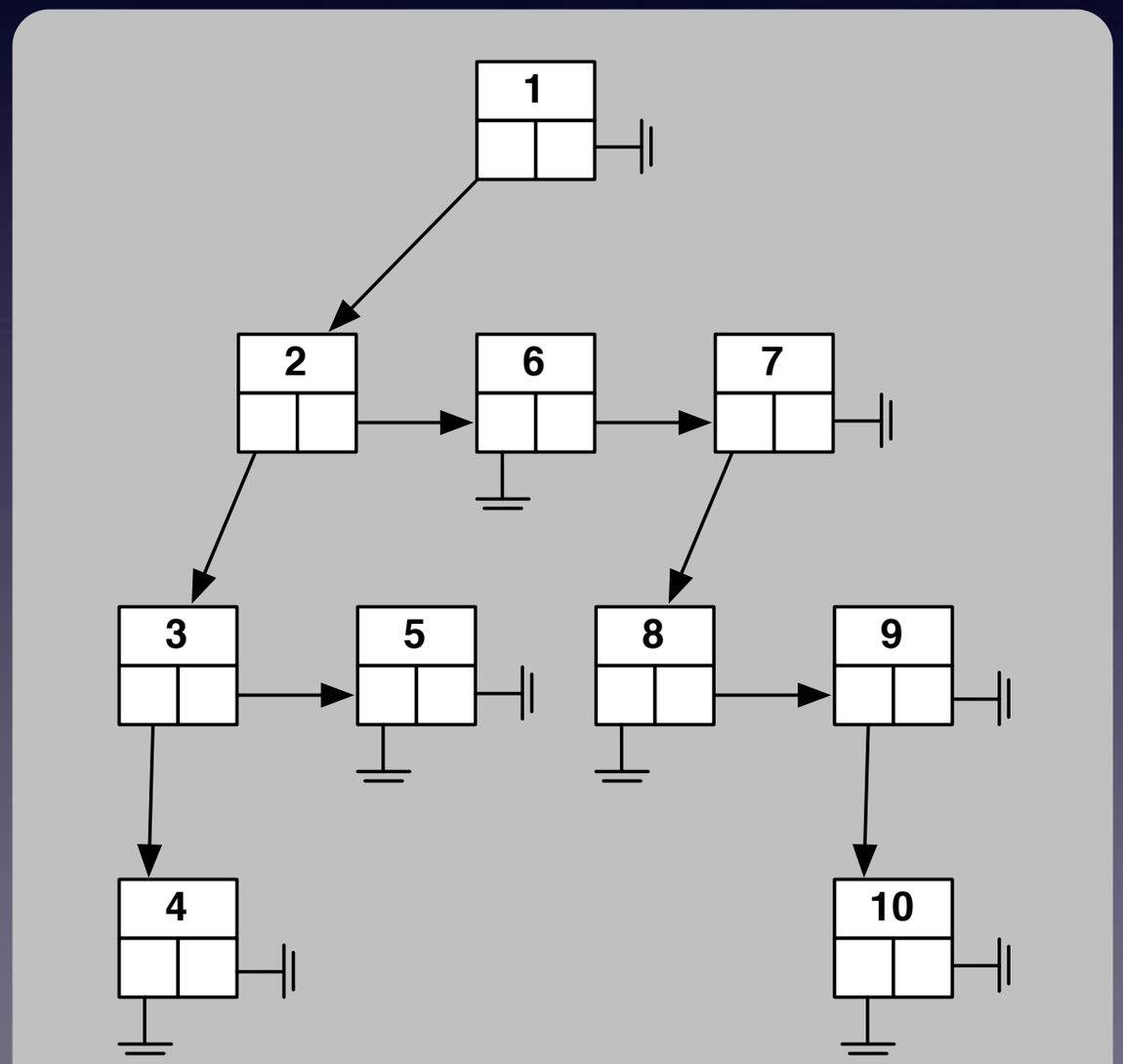
```
Arvore a = new Arvore();
NoArvore n1 = a.criaNo(1);
NoArvore n2 = a.criaNo(2);
NoArvore n3 = a.criaNo(3);
NoArvore n4 = a.criaNo(4);
NoArvore n5 = a.criaNo(5);
NoArvore n6 = a.criaNo(6);
NoArvore n7 = a.criaNo(7);
NoArvore n8 = a.criaNo(8);
NoArvore n9 = a.criaNo(9);
NoArvore n10 = a.criaNo(10);

a.insereFilho(n3, n4);
a.insereFilho(n2, n5);
a.insereFilho(n2, n3);
a.insereFilho(n9, n10);
a.insereFilho(n7, n9);
a.insereFilho(n7, n8);
a.insereFilho(n1, n7);
a.insereFilho(n1, n6);
a.insereFilho(n1, n2);
```



# Implementação

- método toString: imprime o conteúdo dos nós em pré-ordem

```
public String toString() {
        return imprime(raiz);
    }
```



**Algoritmo:**private String imprimeAux(NoArvore no)

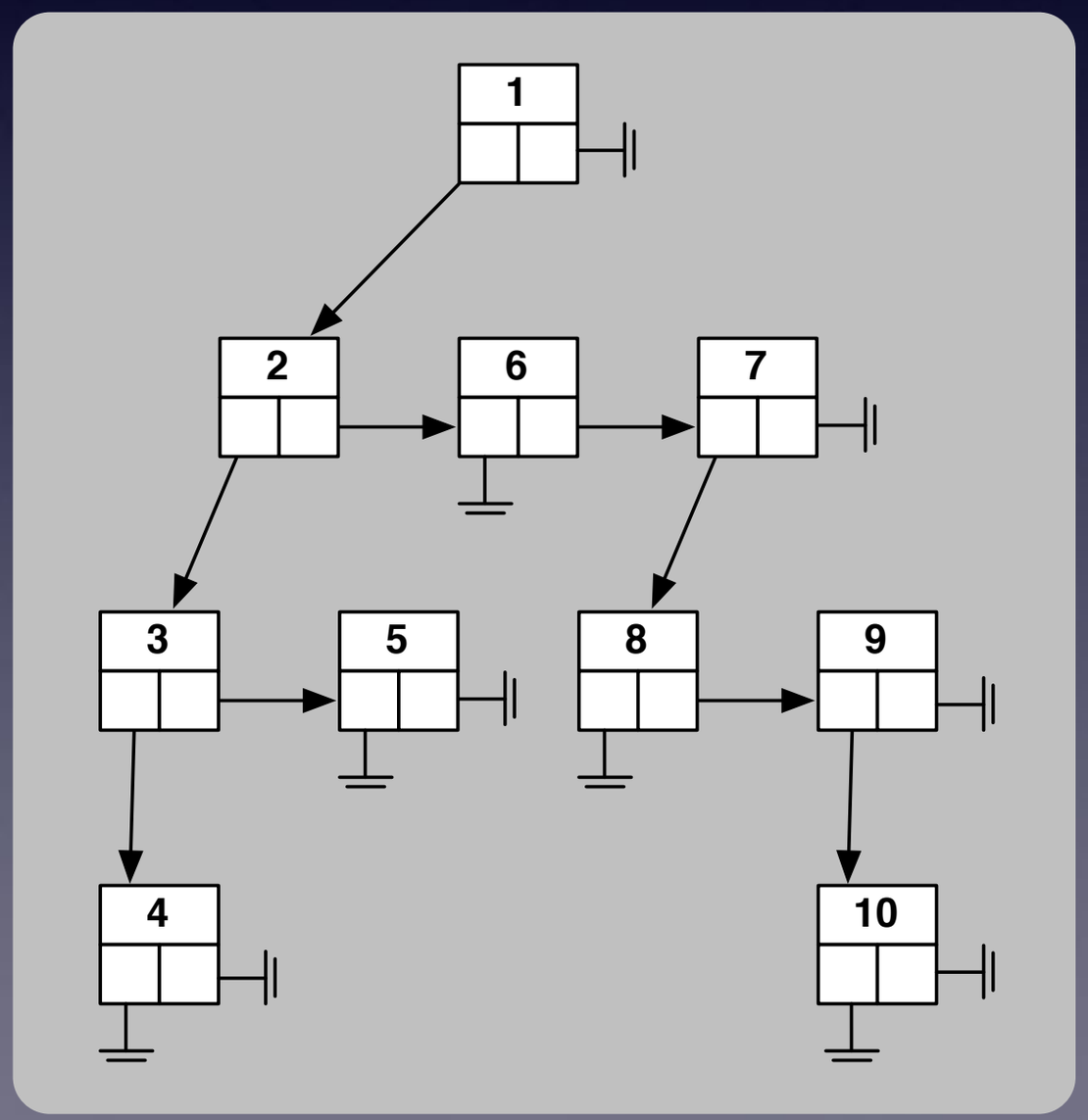
*String s = new String(“”);*
$s \leftarrow s +$ “ < ”;
$s \leftarrow s + no.info$;
*NoArvore* $p \leftarrow$ *no.prim*;
**enquanto** $p \neq null$ **faça**
    $s \leftarrow s + imprimeAux(p)$;
    $p \leftarrow p.prox$;
$s \leftarrow s +$ “ > ”;
**retorna** $s$;

**Algoritmo 4.6**: Método imprimeAux (privado) para Árvores

# Implementação

- método pertence: verifica a existência de uma dada informação na árvore

```
public boolean pertence(int v) {
      return pertence(raiz, v);
   }
```



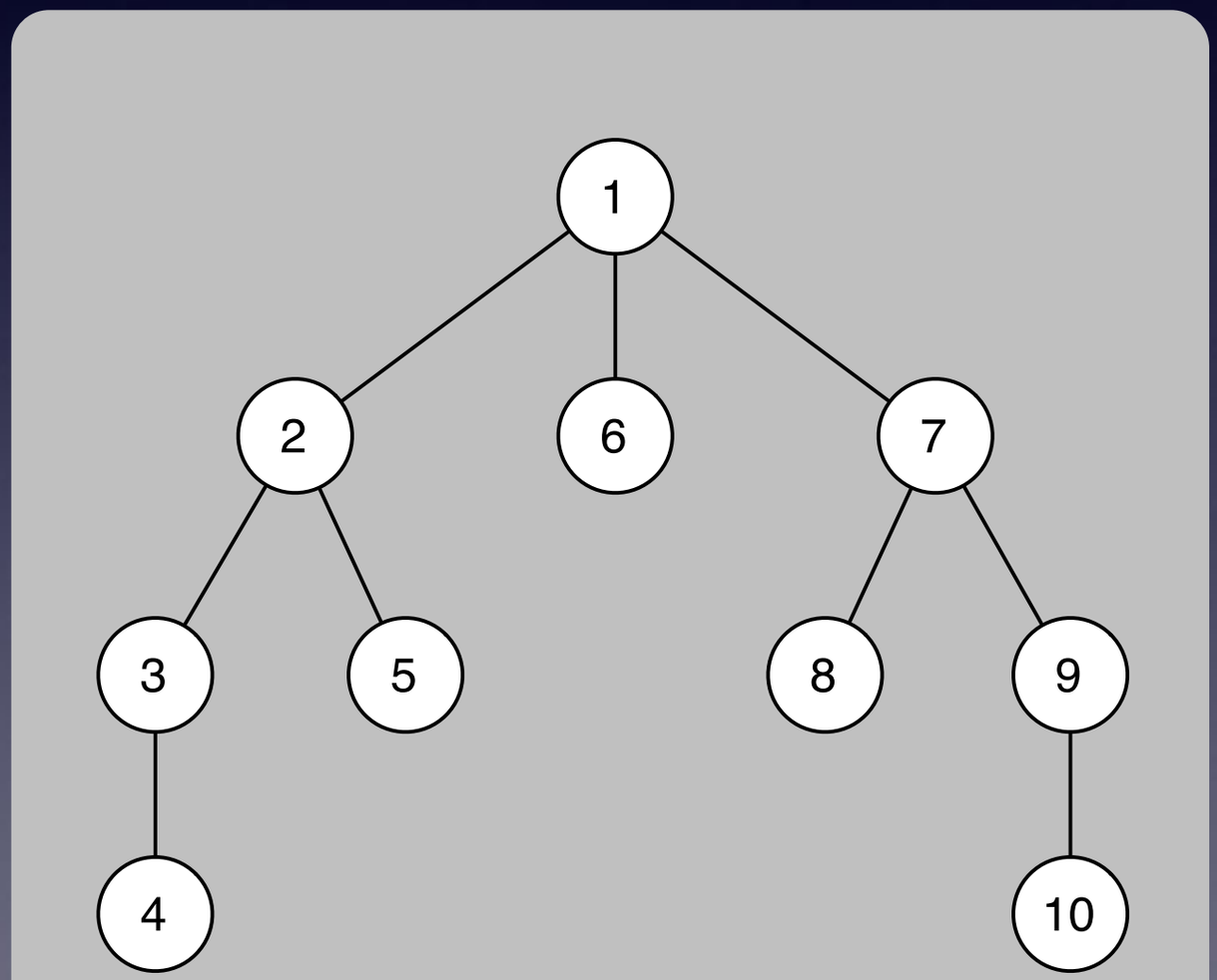
**Algoritmo:**private boolean pertence(NoArvore no, int v)

**se** $(no.info == v)$ **então**
  **retorna** *verdadeiro*;
**senão**
  *NoArvore* $p \leftarrow no.prim$;
  **enquanto** $p \neq null$ **faça**
    **se** $pertence(p, v)$ **então**
      **retorna** *verdadeiro*;
    $p \leftarrow p.prox$;
**retorna** *falso*;

**Algoritmo 4.7**: Método pertence (privado) para Árvores

# Nível e altura

- definidos de forma semelhante a árvores binárias
- Exemplo: h = 3



# Implementação

- método altura:
- maior altura entre as sub-árvores, acrescida de uma unidade

- caso o nó raiz não tenha filhos, a altura da árvore deve ser igual a 0

```
public int altura() {
      return altura(raiz);
   }
```

**Algoritmo:**private int altura(NoArvore no)

```
int hmax ← -1;
NoArvore p ← no.prim;
enquanto p ≠ null faça
    int h ← altura(p);
    se h > hmax então
        hmax ← h;
    p ← p.prox;
retorna hmax + 1;
```

**Algoritmo 4.8**: Método altura (privado) para Árvores

# Resumo

- Árvore binária:
  - uma árvore vazia, ou
  - um nó raiz com duas sub-árvores (sub-árvores da direita SAD e da esquerda SAE)
- Árvore com número variável de filhos:
  - um nó raiz
  - zero ou mais sub-árvores