# Estrutura de Dados – 1º semestre de 2020

Professor Mestre Fabio Pereira da Silva

# Definição

- Listas encadeadas, pilhas e filas são estruturas de dados lineares.
- Uma árvore é uma estrutura de dados não linear, bidimensional, com propriedades especiais.

# Árvores

- Árvores são utilizadas para realizar a representação dos elementos de um determinado conjunto de dados de maneira hierárquica.
  - Representação de uma hierarquia de pastas
  - Diagrama hierárquico de uma organização
  - Modelagem de algoritmos
- O conceito de árvore está diretamente ligado à recursão.

# Definição de Árvores

- Árvores são um conjunto finito de elementos.
  - Um elemento é chamado de **raíz**
- Os outros são divididos em subconjuntos disjuntos onde cada um define uma árvore.
  - Cada elemento é um **Nó ou Vértice da árvore**
  - Arcos ou arestas conectam os vértices

# Definição de Árvores

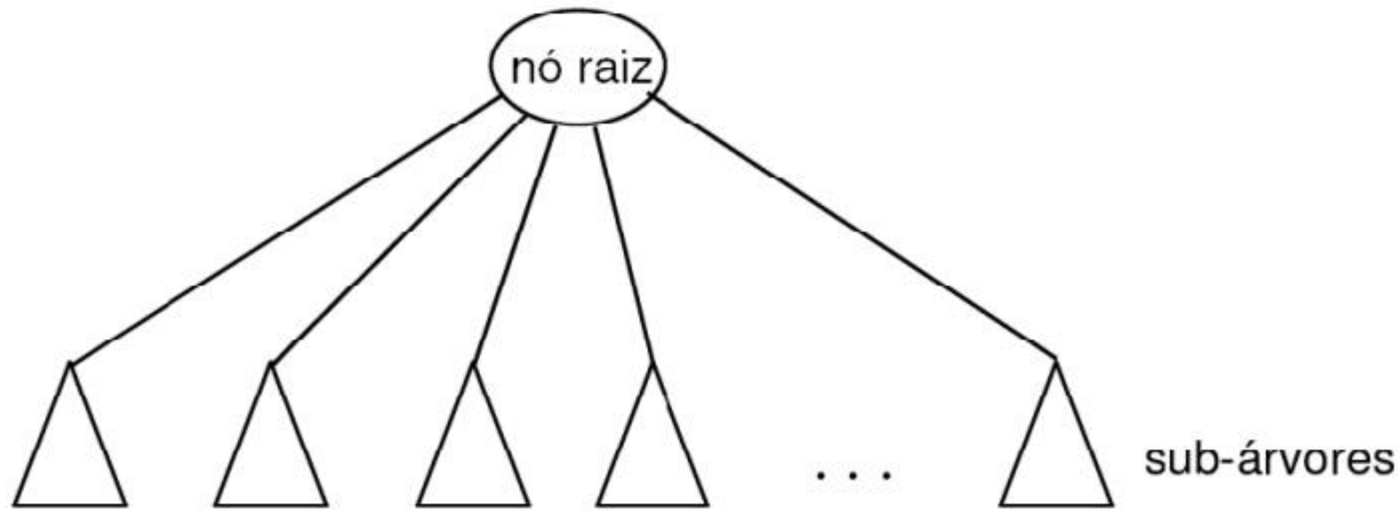
- Uma coleção não vazia de vértices e ramos que satisfazem a certos requisitos.
- Vértice (Ou Nó)
  - É um objeto simples que pode ter um nome e mais alguma outra informação associada.
- Arco ou aresta (direcionado ou não)
  - É a conexão entre dois Nós

# Definição de Árvores

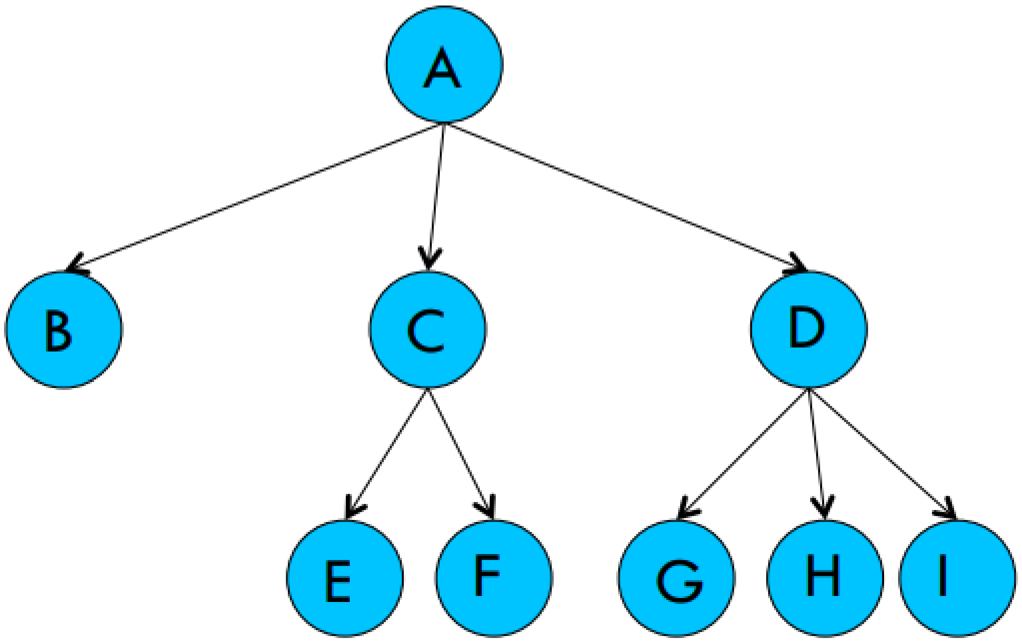
- Nós filhos, pais, tios, irmãos e avô
- Grau de saída (número de filhos de um nó)
- Nó folha (grau de saída nulo) e nó interior (grau de saída diferente de nulo)
- Grau de uma árvore (máximo grau de saída)
- Floresta (conjunto de árvores)

# Definição de Árvores

- 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 da árvore são os nós sem filhos



# Representação de árvore



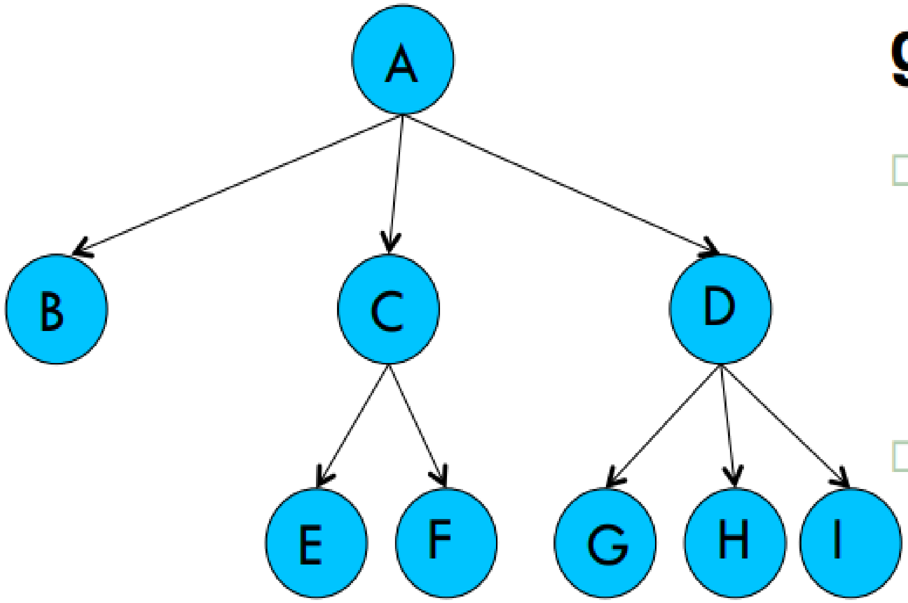
# Terminologia

- Cada vértice (exceto a raiz) tem exatamente um **antecessor imediato ou pai.**
- Cada vértice tem **nós sucessores imediatos ou filhos**.
- Nós sem filhos são considerados **nós terminais ou folhas.**
- Filhos de um mesmo pai: **irmãos**
- Nós com pelo menos um filho: **não terminais ou internos.**

# Terminologia

- Caminho em uma árvore
  - É uma lista de vértices distintos e sucessivos, conectados por arcos (arestas) da árvore
- Nó raiz
  - Existe exatamente um caminho entre a raiz e cada um dos nós da árvore
- Qualquer nó é a raiz de uma sub-árvore consistindo dele e dos Nós abaixo
- Se existir mais de um caminho ou nenhum: **grafo**

# Terminologia




**grau de um vértice**

- é o número de subárvores não vazias de um nó
- no exemplo
  - grau de A = 3
  - grau de C = 2

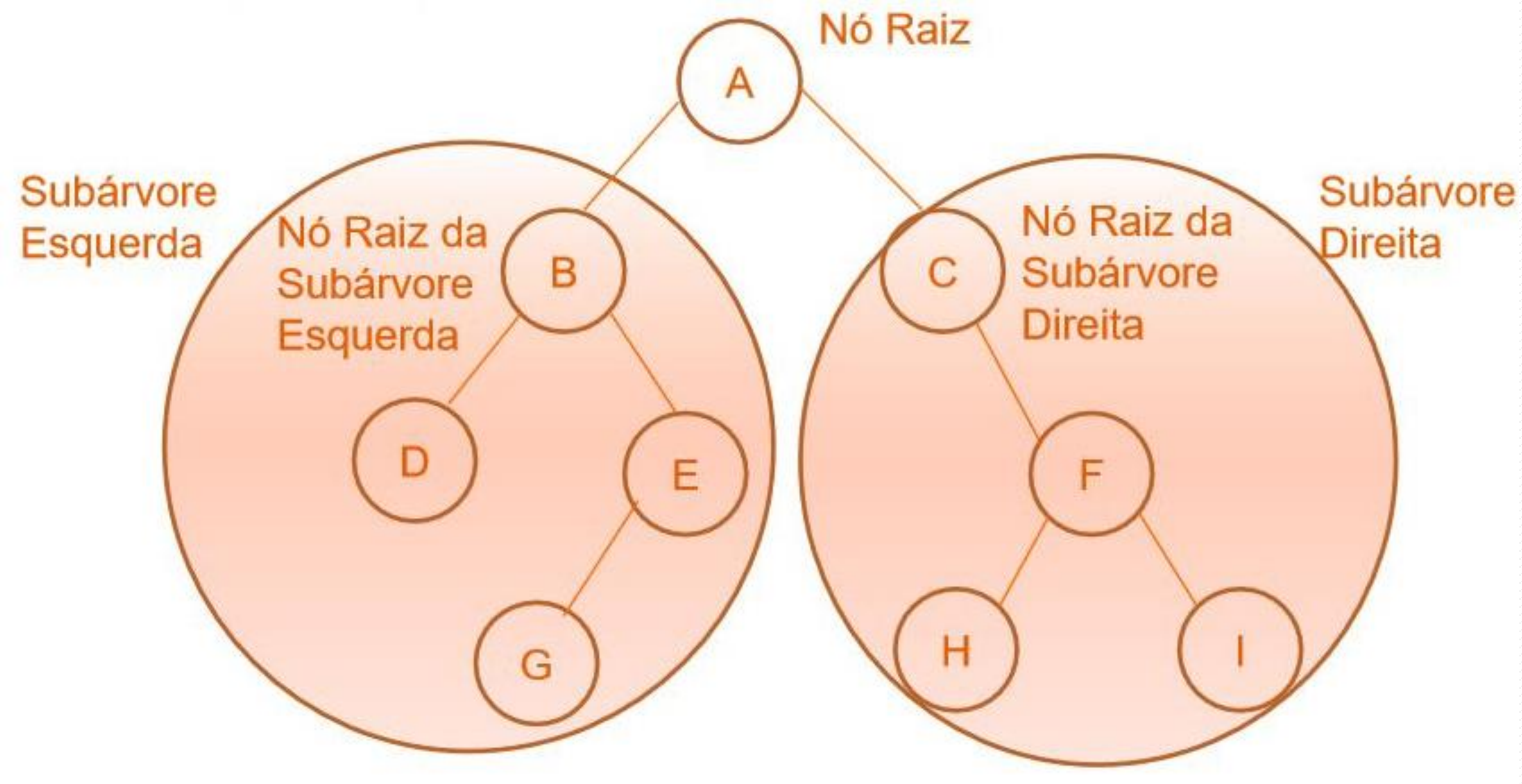
# Terminologia

- Altura de uma árvore
  - Maior distância entre a raiz e qualquer nó
- Floresta
  - Um conjunto de árvores
  - Se removermos a raiz e os arcos que ligam às sub-árvores, ficamos com uma floresta

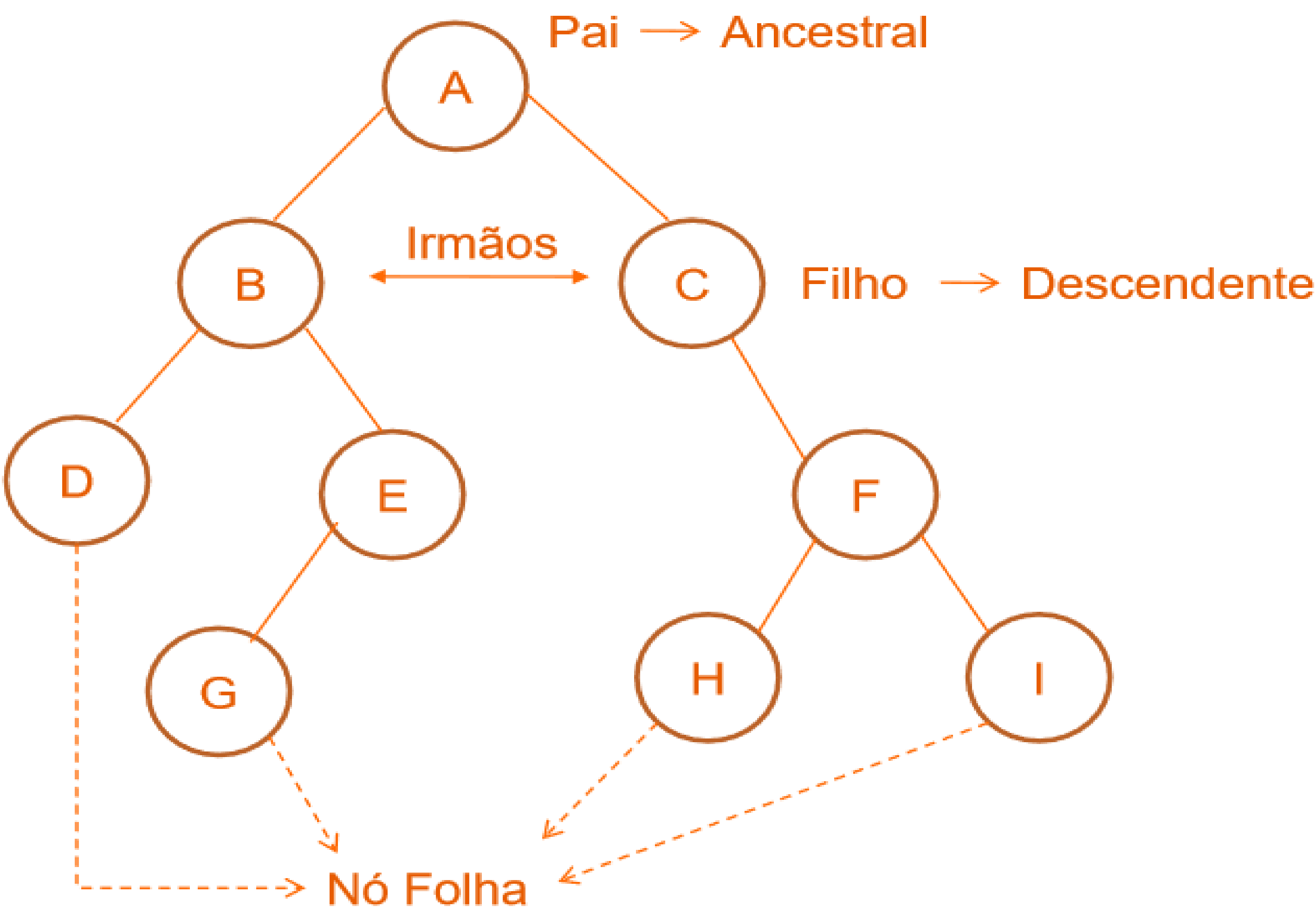
# Árvores Binárias

- É um conjunto finito de elementos que é vazio ou composto de três conjuntos disjuntos
- O primeiro contém um único elemento, a raiz
- Os outros dois subconjuntos são árvores binárias
  - As sub-árvores da esquerda e da direita
  - As sub-árvores da esquerda ou da direita podem estar vazias

# Árvores Binárias

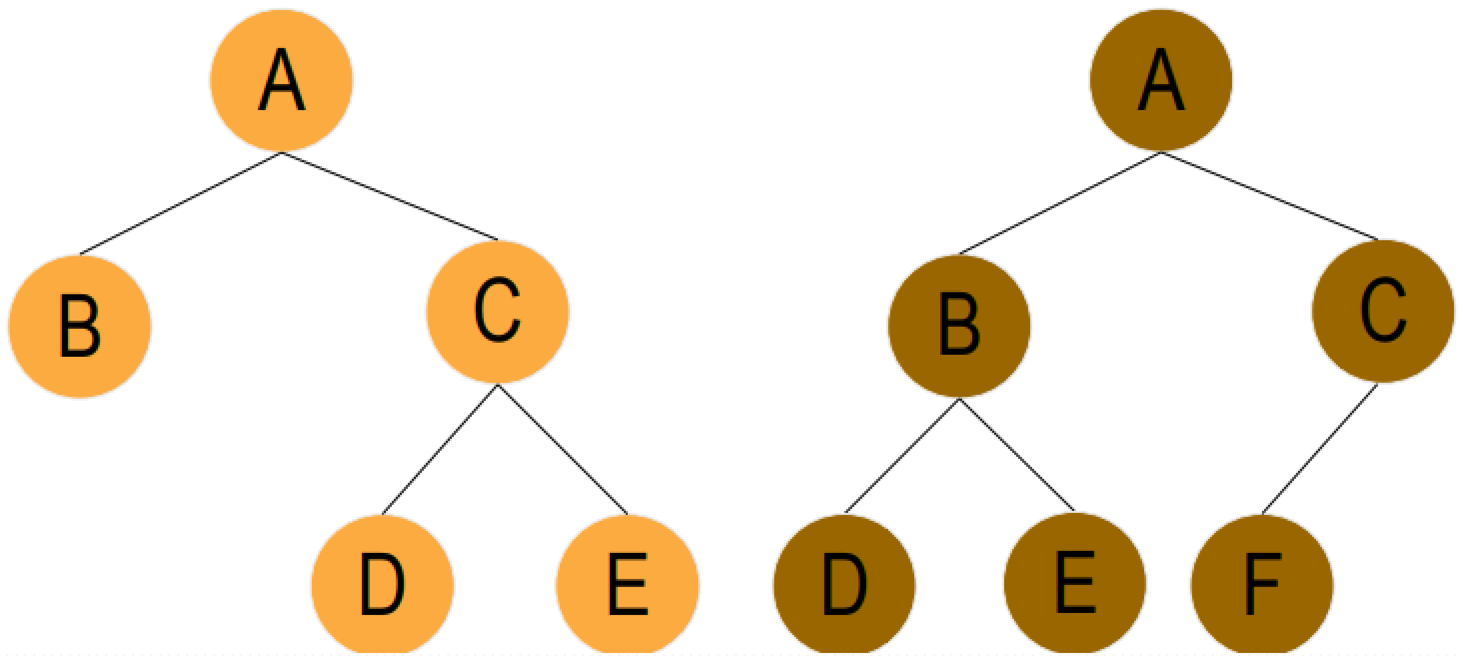


# Árvores Binárias



# Árvore Binária Balanceada

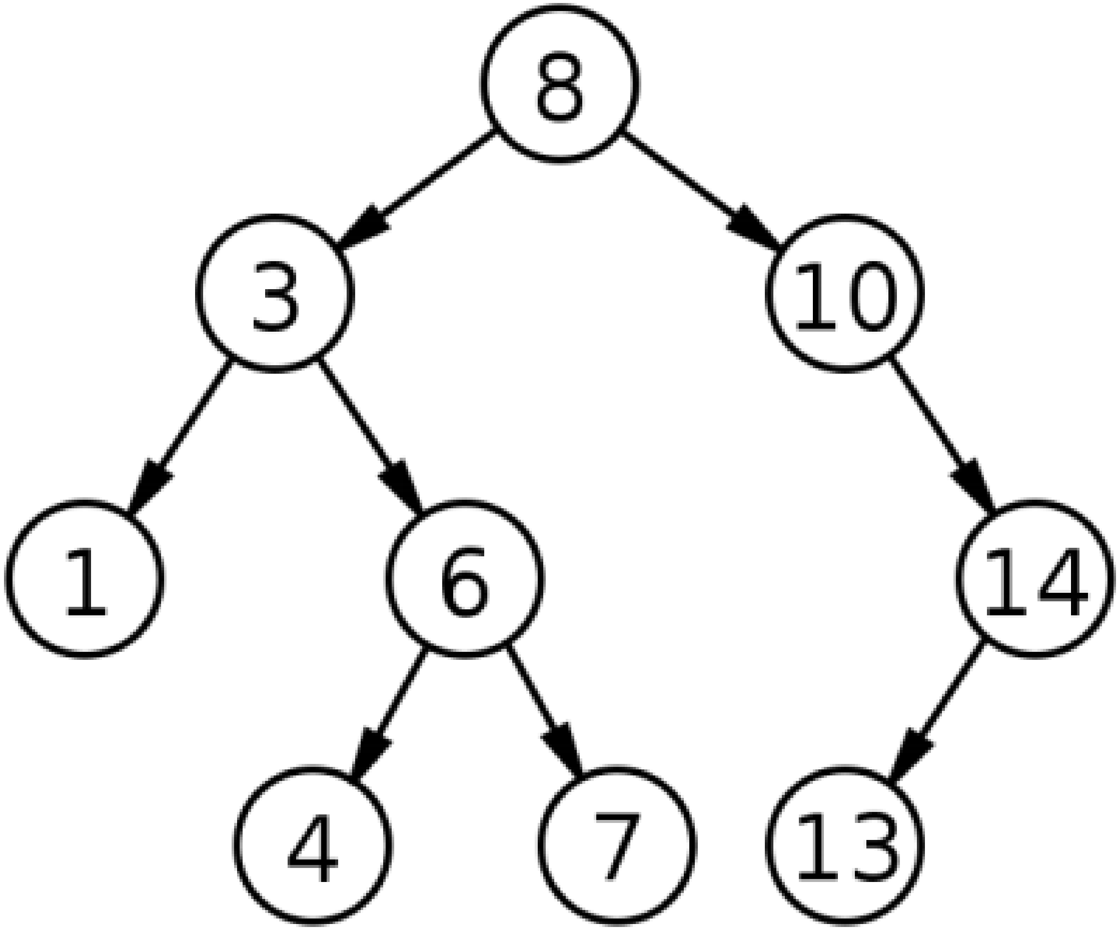
- Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1



# Árvore Binária Balanceada

- Pior caso: como o número de passos é determinado pela altura da árvore, o pior caso é a árvore degenerada (altura = n).
  - Altura da árvore depende da sequência de inserção das chaves
  - Considere, p.ex., o que acontece se uma sequência ordenada de chaves é inserida
- Busca ótima: árvore de altura mínima (perfeitamente balanceada)
- Busca eficiente: árvore razoavelmente balanceada...(árvore balanceada)

# Exemplo de Árvore Binária

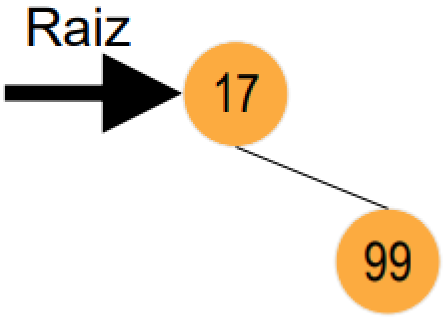


# Inserção em Árvores Binárias

- Procure um “local” para inserir a nova chave, começando a procura a partir do nó-raiz:
- Para cada nó-raiz, compare:
  - Se a nova chave for menor do que o valor no nó-raiz, repita o processo para sub-árvore esquerda; ou
  - Se a nova chave for maior que o valor no nó-raiz, repita o processo para sub-árvore direita.
- Se um ponteiro (filho esquerdo/direito de um nó-raiz) nulo é atingido, coloque o novo nó como sendo raiz dessa sub-árvore vazia.
- **A inserção sempre se dá como nó folha: não exige deslocamentos**

# Inserção em Árvores Binárias

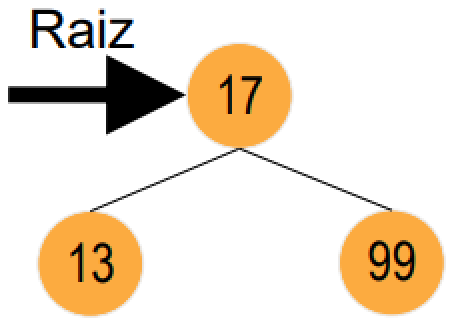
- **Conjunto:** [17, 99, 13, 1, 3, 100, 400]
  - O número 17 será inserido tornando-se o nó raiz
  - A inserção do 99 inicia-se na raiz. Compara-se 99 c/ 17.
  - Como 99 > 17, 99 deve ser colocado na sub-árvore direita do nó contendo 17 (subárvore direita, inicialmente, nula)



# Inserção em Árvores Binárias

- **Conjunto:** [17, 99, 13, 1, 3, 100, 400]

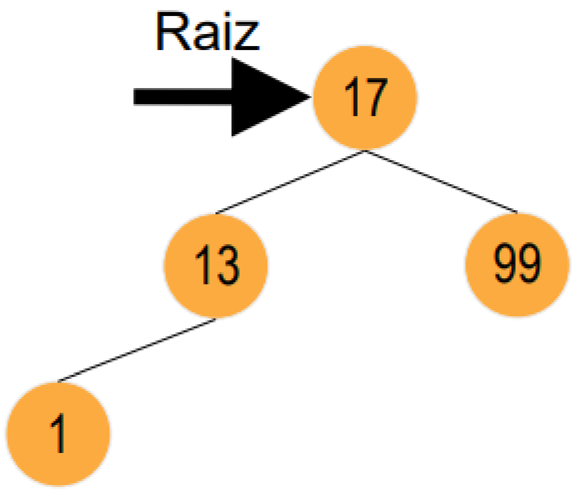
- A inserção do 13 inicia-se na raiz
- Compara-se 13 c/ 17. Como 13 < 17, 13 deve ser colocado na sub-árvore esquerda do nó contendo 17
- Já que o nó 17 não possui descendente esquerdo, 13 é inserido como raiz dessa sub-árvore



# Inserção em Árvores Binárias

- **Conjunto:** [17, 99, 13, 1, 3, 100, 400]

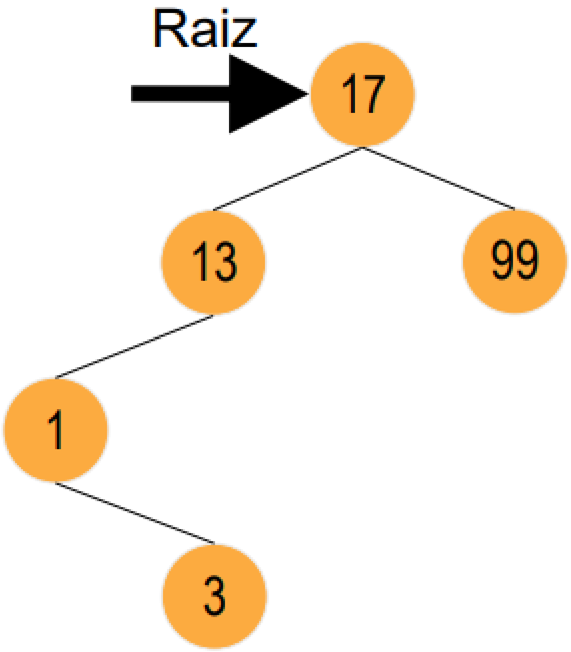
- Repete-se o procedimento para inserir o valor 1
- 1<17, então será inserido na sub-árvore esquerda
- Chegando nela, encontra-se o nó 13, 1<13 então ele será inserido na sub-árvore esquerda de 13



# Inserção em Árvores Binárias

- **Conjunto: [17, 99, 13, 1, 3, 100, 400]**

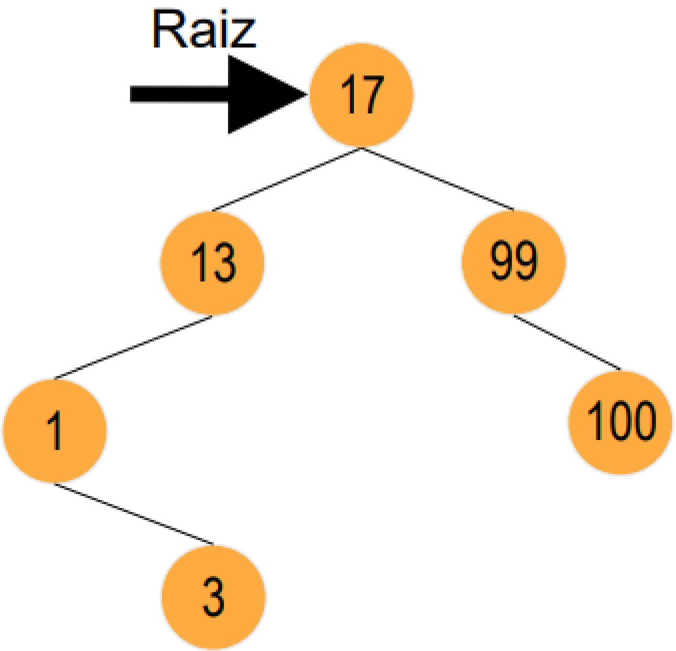
- Repete-se o procedimento para inserir o elemento 3:
  - 3 < 17;
  - 3 < 13
  - 3 > 1



# Inserção em Árvores Binárias

- **Conjunto:** [17, 99, 13, 1, 3, 100, 400]

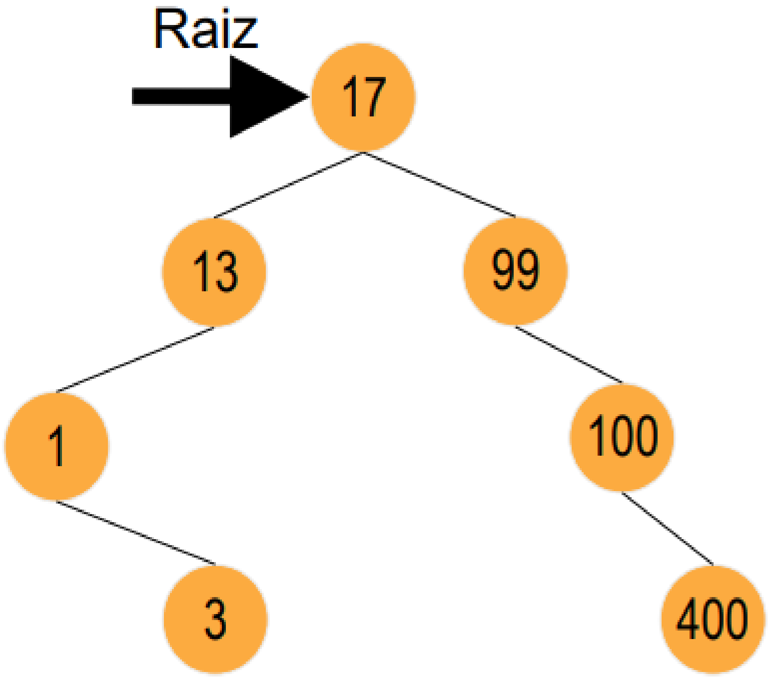
- Repete-se o procedimento para inserir o elemento 100:
  - 100 > 17
  - 100 > 99



# Inserção em Árvores Binárias

- **Conjunto: [17, 99, 13, 1, 3, 100, 400]**

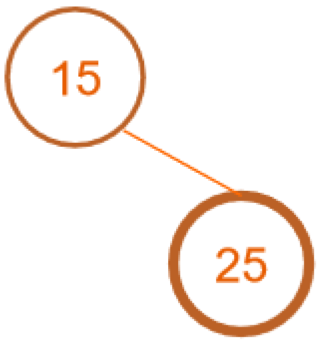
- Repete-se o procedimento para inserir o elemento 400:
  - 400 > 17
  - 400 > 99
  - 400 > 100



# Inserção em Árvores Binárias

.

**25**, 5, 30, 8, 20, 31.



# Inserção em Árvores Binárias

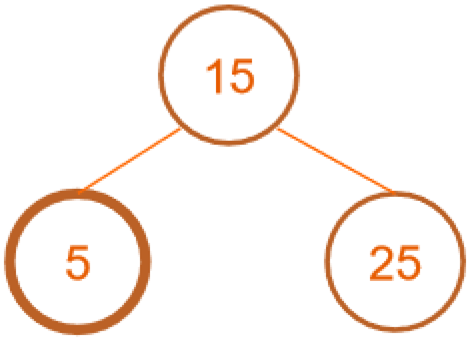
.

**15**, 25, 5, 30, 8, 20, 31.

# Inserção em Árvores Binárias

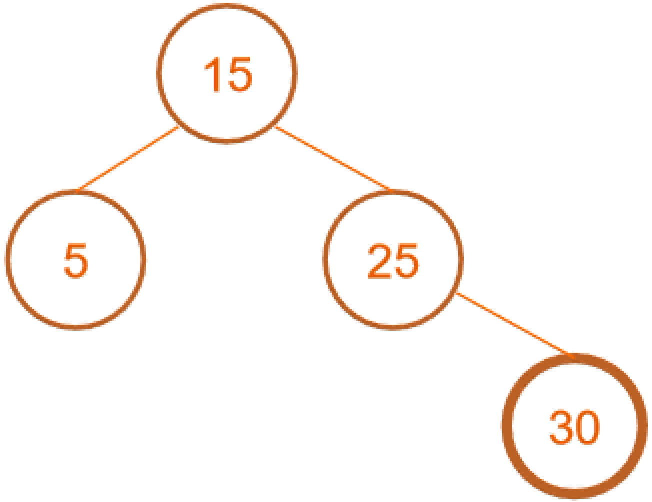
.

**5**, 30, 8, 20, 31.



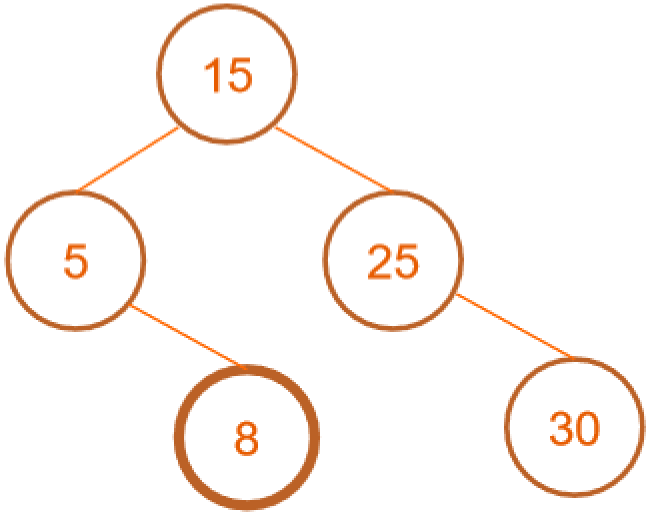
# Inserção em Árvores Binárias

**30**, 8, 20, 31.



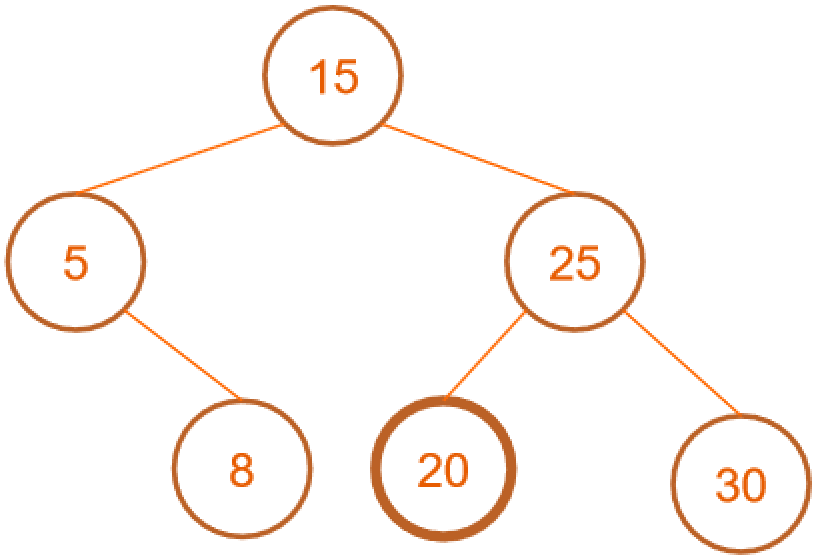
# Inserção em Árvores Binárias

**8**, 20, 31.



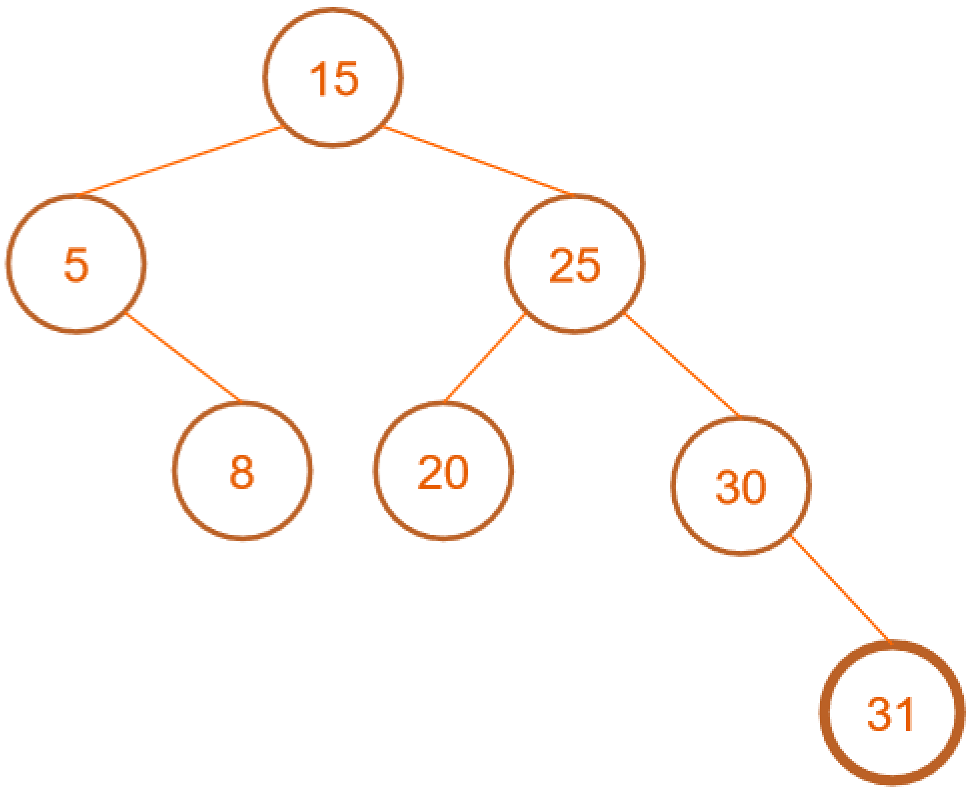
# Inserção em Árvores Binárias

**20**, 31.



# Inserção em Árvores Binárias

**31**.



# Método de inserção em Árvore Binária

```
public void adicionaElemento(int e){
    No novo = new No(e);
    No aux1=raiz; No aux2=raiz;
    if (aux1 != null){
        while (aux1 != null && aux1.dados != e){
            aux2=aux1;
            if (e<aux1.dados)
                aux1=aux1.esquerda;
            else if (e>aux1.dados)
                aux1=aux1.direita;
        }
        if (e == aux2.dados)
            System.out.println("Elemento já existe");
        else{
            if (e < aux2.dados)
                aux2.esquerda = novo;
            if (e > aux2.dados)
                aux2.direita = novo;
            System.out.println(e+" Incluído");
        }
    }
    else{
         raiz=novo;
         System.out.println(e+" Incluído");
        }
}
```

# Custo de inserção em Árvores Binárias

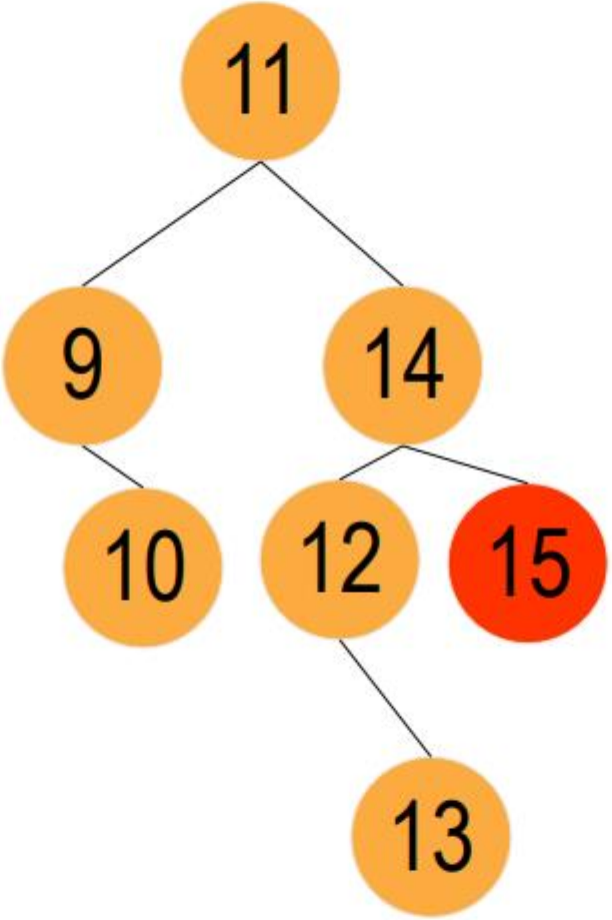
- A inserção requer uma busca pelo lugar da chave, portanto, com custo de uma busca qualquer (tempo proporcional à altura da árvore).
- O custo da inserção, após a localização do lugar, é constante; não depende do número de nós.
- Logo, tem complexidade análoga à da busca

# Remoção em Árvores Binárias

- Casos a serem considerados no algoritmo de remoção de nós de uma árvore binária:
- **Caso 1: o nó é folha:**
  - O nó pode ser retirado sem problema;
- **Caso 2: o nó possui uma sub-árvore (esq./dir.)**
  - O nó-raiz da sub-árvore (esq./dir.) pode substituir o nó eliminado;
- **Caso 3: o nó possui duas sub-árvores**
  - O nó cuja chave seja a menor da sub-árvore direita pode substituir o nó eliminado; ou, alternativamente, o de maior valor da sub-árvore esquerda pode substituí-lo.

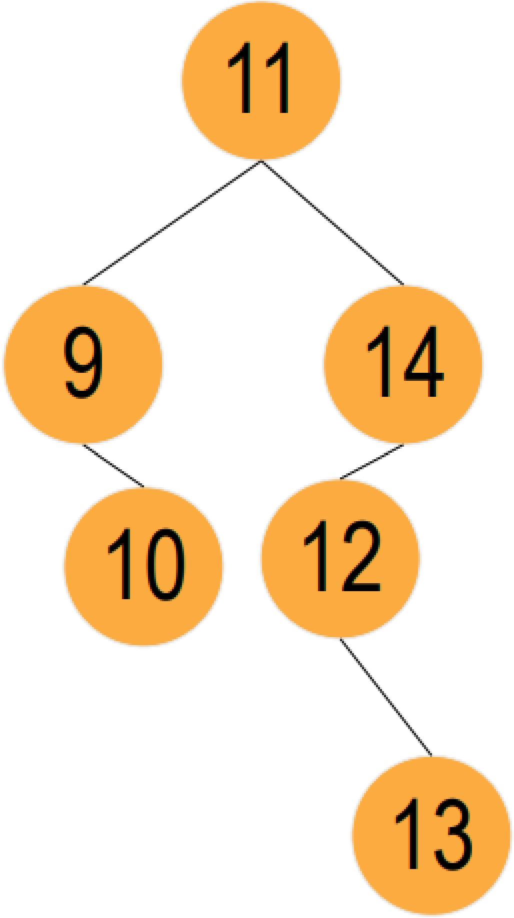
# Remoção em Árvores Binárias – Caso 1

- Caso o valor a ser removido seja o 15
- pode ser removido sem problema, não requer ajustes posteriores



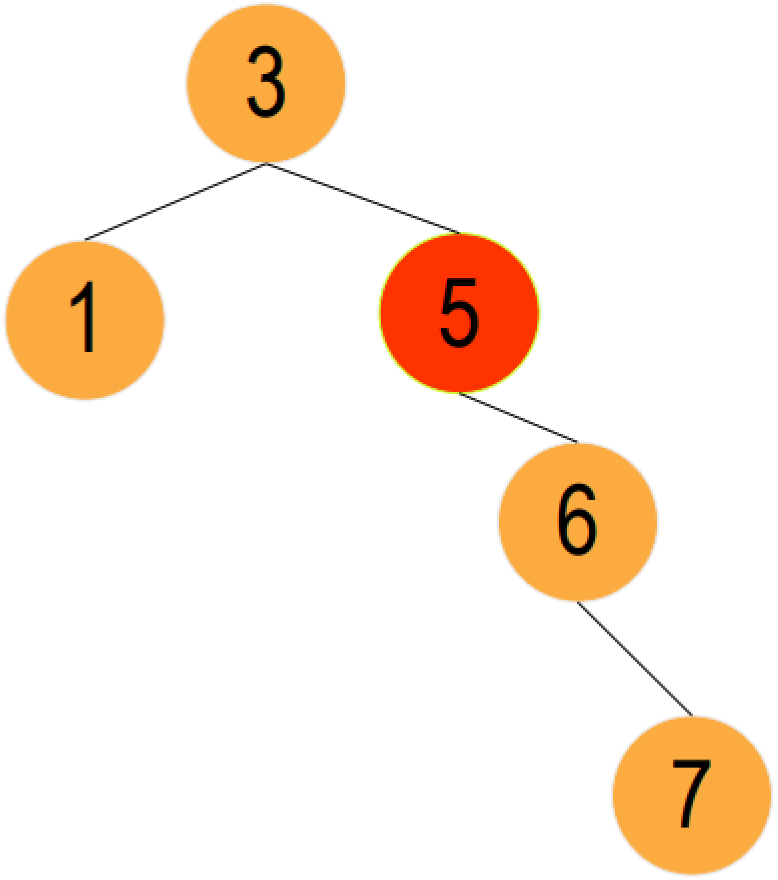
# Remoção em Árvores Binárias – Caso 1

- Os nós com os valores 10 e 13 também podem ser removidos!



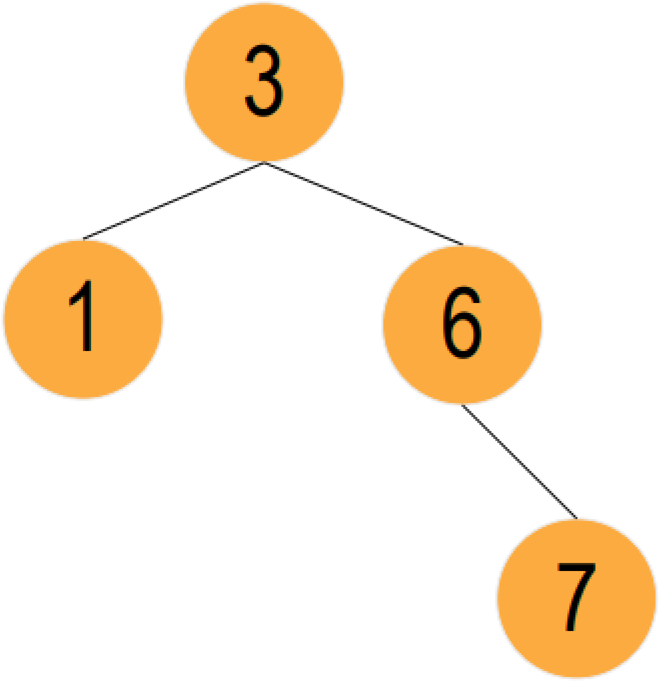
# Remoção em Árvores Binárias – Caso 2

- Removendo-se o nó com o valor 5
- Como ele possui uma sub-árvore direita, o nó contendo o valor 6 pode "ocupar" o lugar do nó removido



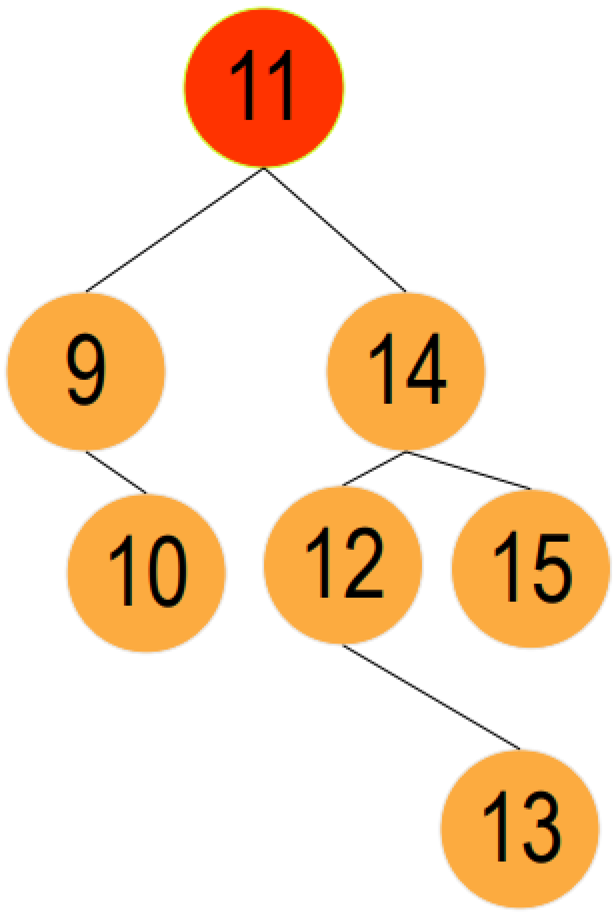
# Remoção em Árvores Binárias – Caso 2

- Caso existisse um nó com somente uma sub-árvore esquerda, seria análogo.



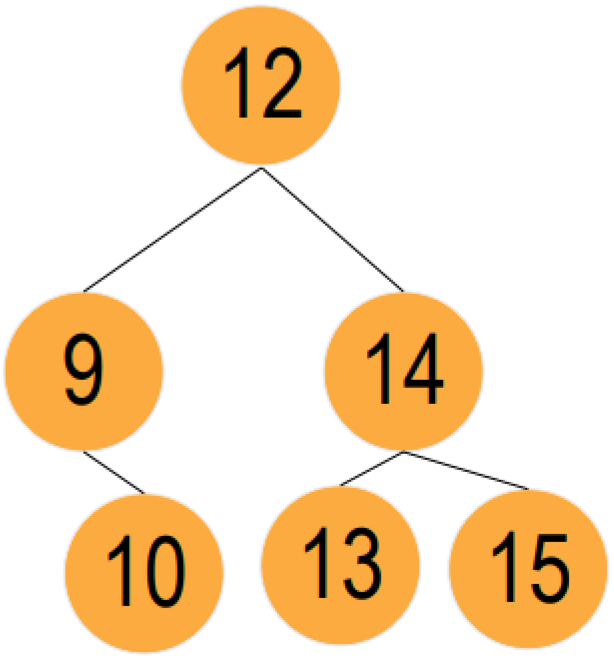
# Remoção em Árvores Binárias – Caso 3

- Eliminando-se o nó de chave 11
- Neste caso, existem 2 opções:
  - O nó com chave 10 pode “ocupar” o lugar do nó-raiz, ou
  - O nó com chave 12 pode “ocupar” o lugar do nó-raiz



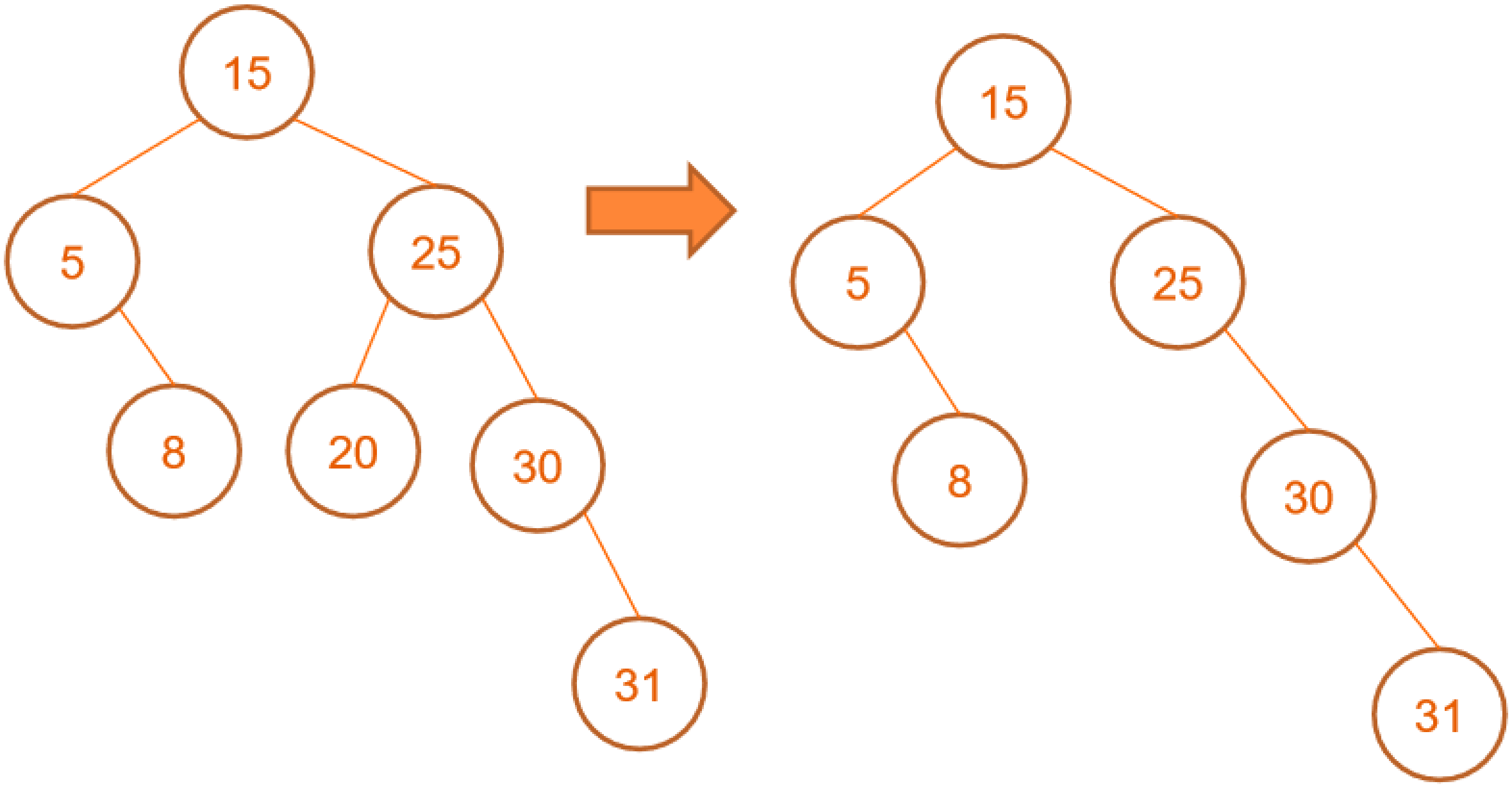
# Remoção em Árvores Binárias – Caso 3

- Esse terceiro caso, também se aplica ao nó com chave 14, caso seja retirado.
  - Nessa configuração, os nós com chave 13 ou 15 poderiam ocupar seu lugar.



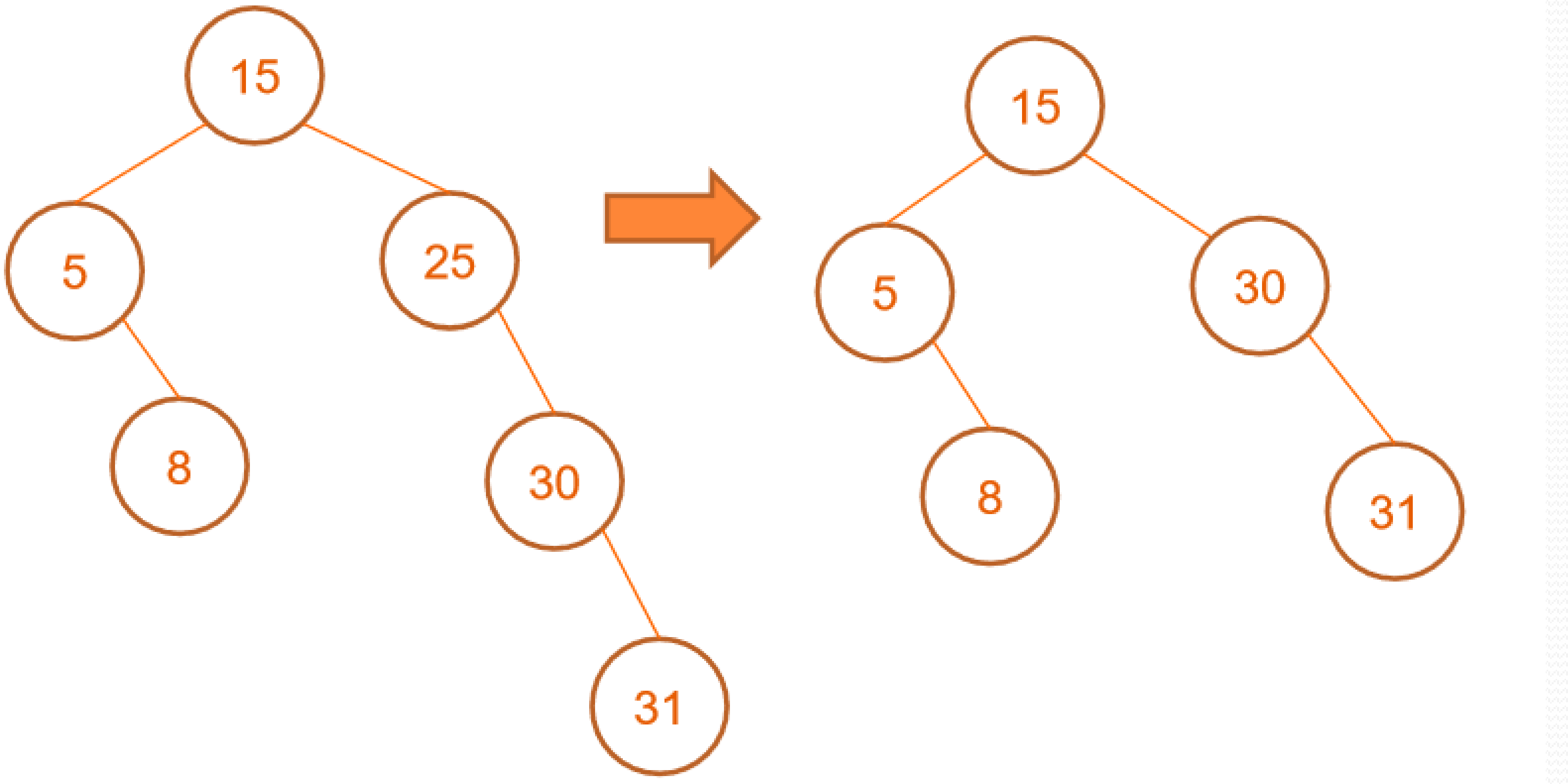
# Remoção em Árvores Binárias

Remover o nó com o elemento 20



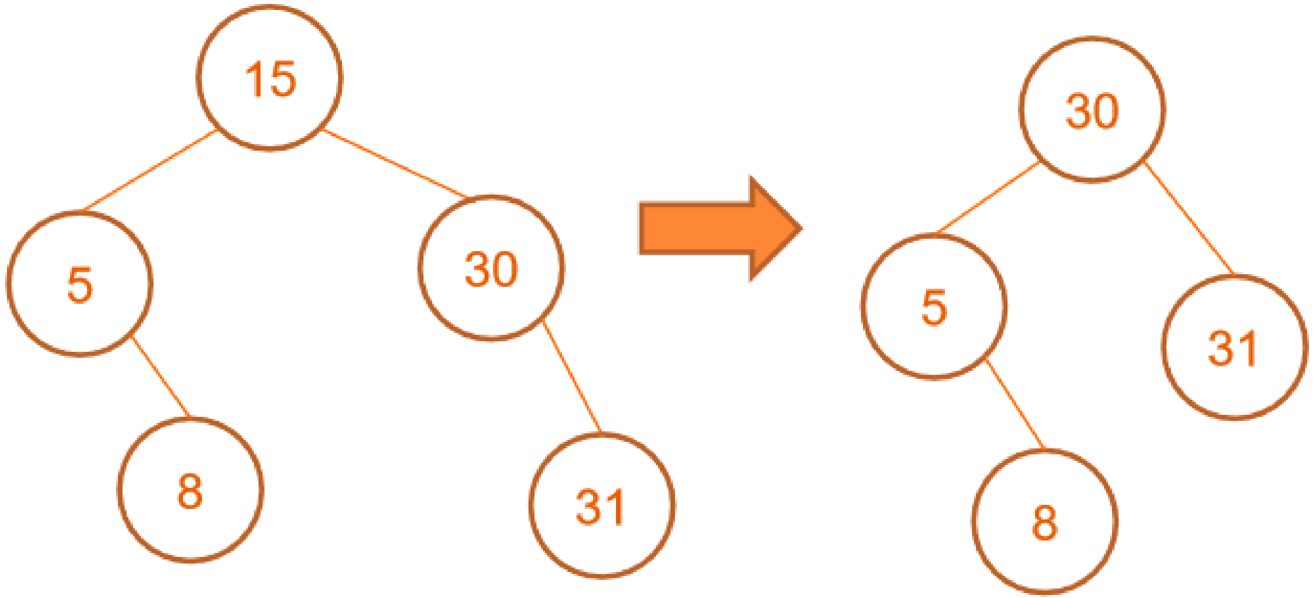
# Remoção em Árvores Binárias

Remover o nó com o elemento 25



# Remoção em Árvores Binárias

Remover o nó com o elemento 15



# Custo de remoção em Árvores Binárias

- A remoção requer uma busca pela chave do nó a ser removido, portanto, com custo de uma busca qualquer (tempo proporcional à altura da árvore).
- O custo da remoção, após a localização do nó dependerá de 2 fatores:
  - do caso em que se enquadra a remoção: se o nó tem 0, 1 ou 2 sub-árvores; se 0 ou 1 filho, custo é constante.
  - de sua posição na árvore, caso tenha 2 sub-árvores (quanto mais próximo do último elemento, menor esse custo)
- Repare que um maior custo na busca implica num menor custo na remoção pp. dita; e vice-versa.
- Logo, tem complexidade dependente da altura da árvore.

# Percurso em Pré-Ordem

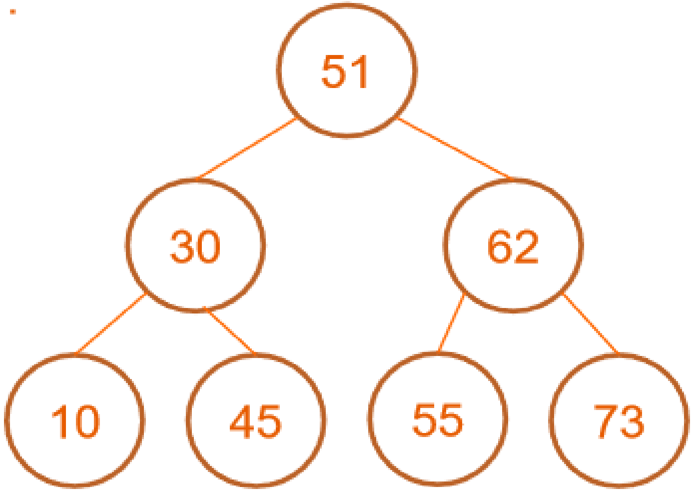
Percurso em Profundidade (Pré-Ordem)

1º. Visitar a raiz (mostrar elemento)

2º. Percorrer a subárvore esquerda

3º. Percorrer a subárvore direita

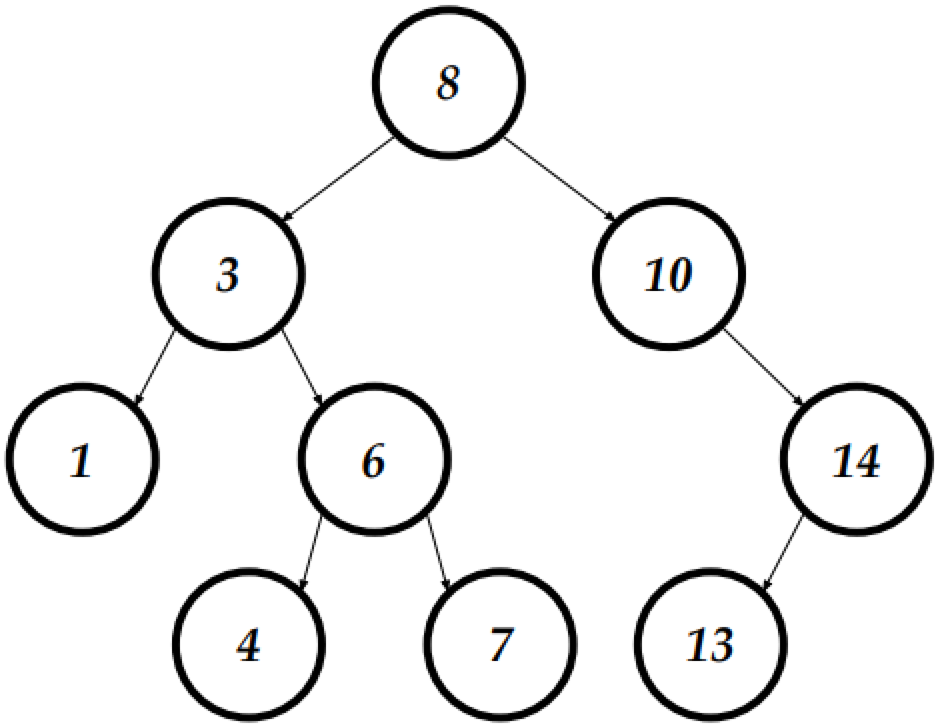
Considere a árvore a seguir, mostre a seqüência gerada com percurso em profundidade :




Solução: 51, 30, 10, 45, 62, 55, 73

# Percurso em Pré-Ordem

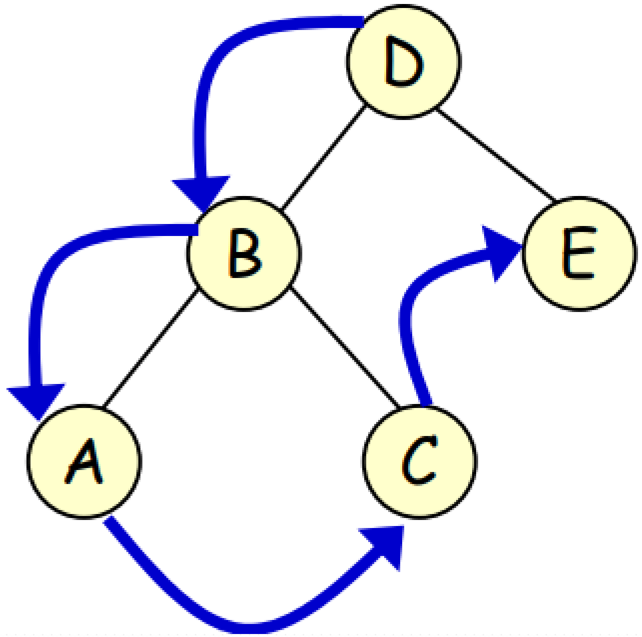
Pre Ordem: 8, 3, 1, 6, 4, 7, 10, 14, 13




```
void printPreOrder(node_t *node) {
  if (node == NULL) {
    return;
  }
  printf("%d ", node->value);
  printPreOrder(node->leftChild);
  printPreOrder(node->rightChild);
}
```

# Percurso em Pré-Ordem

.




D B A C E

# Percurso em Ordem

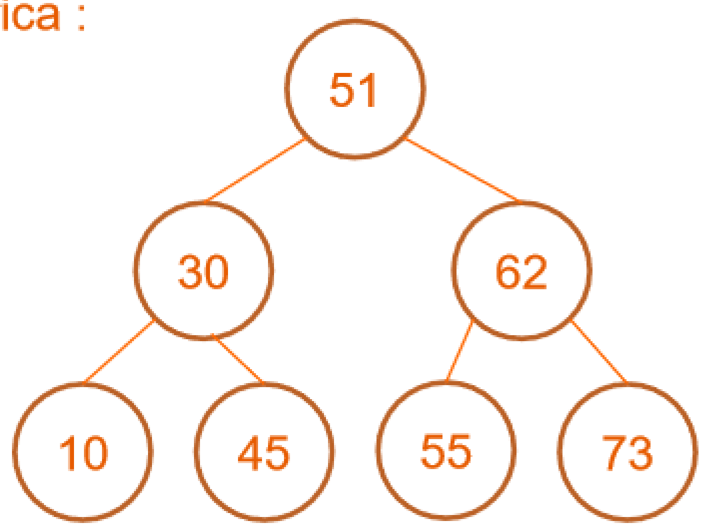
Percurso em Ordem Simétrica (em Ordem)

1º. Percorrer a subárvore esquerda

2º. Visitar a raiz (mostrar elemento)

3º. Percorrer a subárvore direita

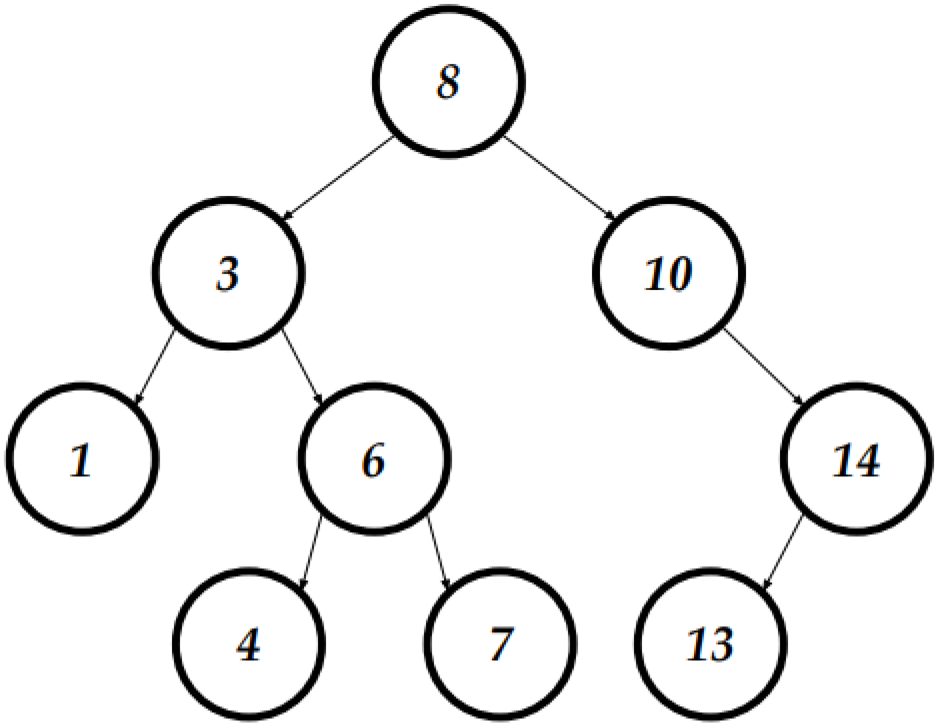
Considere a árvore a seguir, mostre a seqüência gerada com percurso em ordem simétrica :




Solução: 10, 30, 45, 51, 55, 62, 73

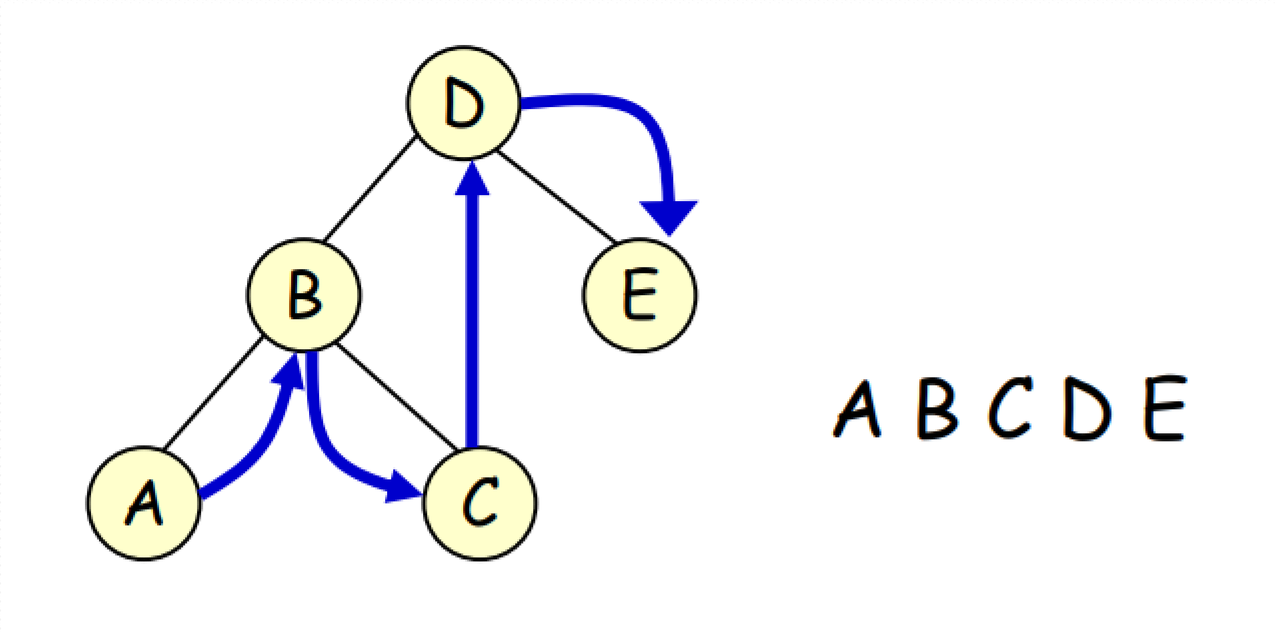
# Percurso em Ordem

Em Ordem: 1, 3, 4, 6, 7, 8, 10, 13, 14




```c
void printInOrder(node_t *node) {
  if (node == NULL) {
    return;
  }
  printInOrder(node->leftChild);
  printf("%d ", node->value);
  printInOrder(node->rightChild);
}
```

# Percurso em Ordem

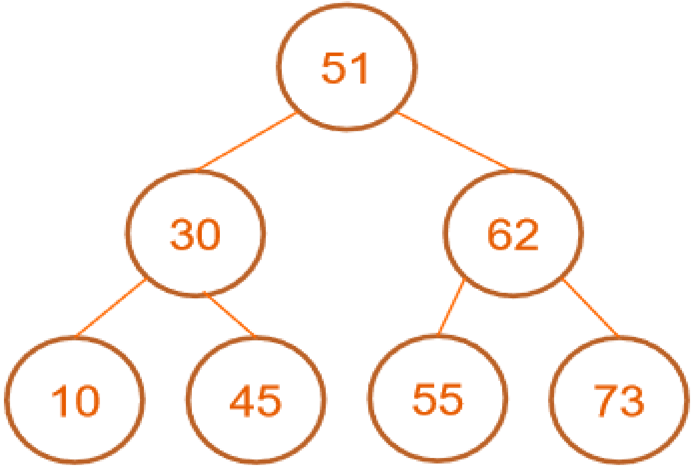


# Percurso em Pós-Ordem

Percurso em Pós Ordem

1º. Percorrer a subárvore esquerda
2º. Percorrer a subárvore direita
3º. Visitar a raiz (mostrar elemento)

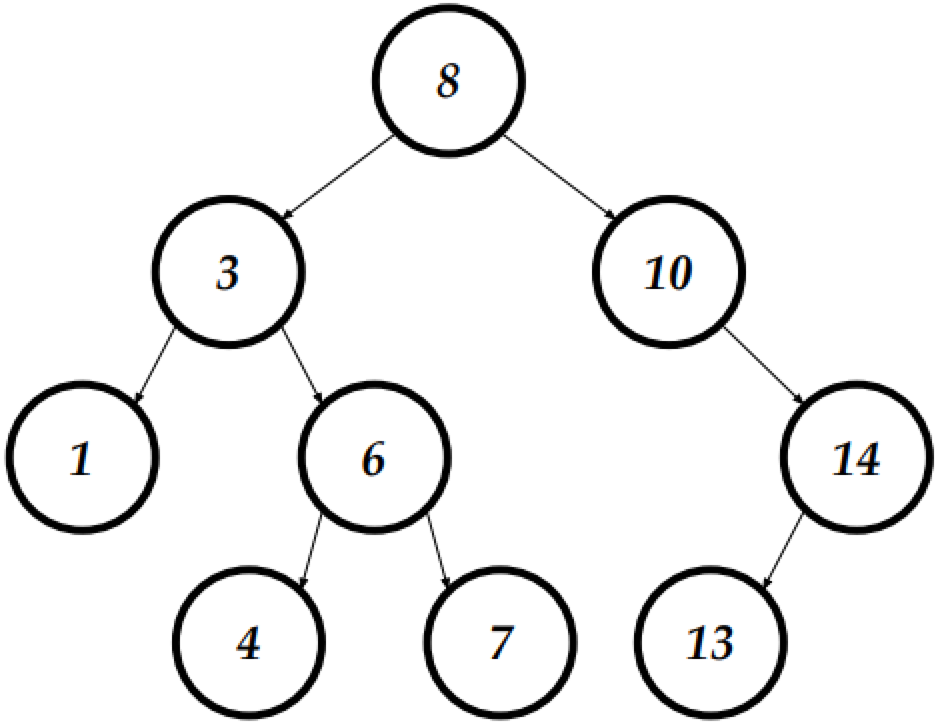
Considere a árvore a seguir, mostre a seqüência gerada com percurso em pós ordem:




Solução: 10, 45, 30, 55, 73, 62, 51

# Percurso em Pós-Ordem

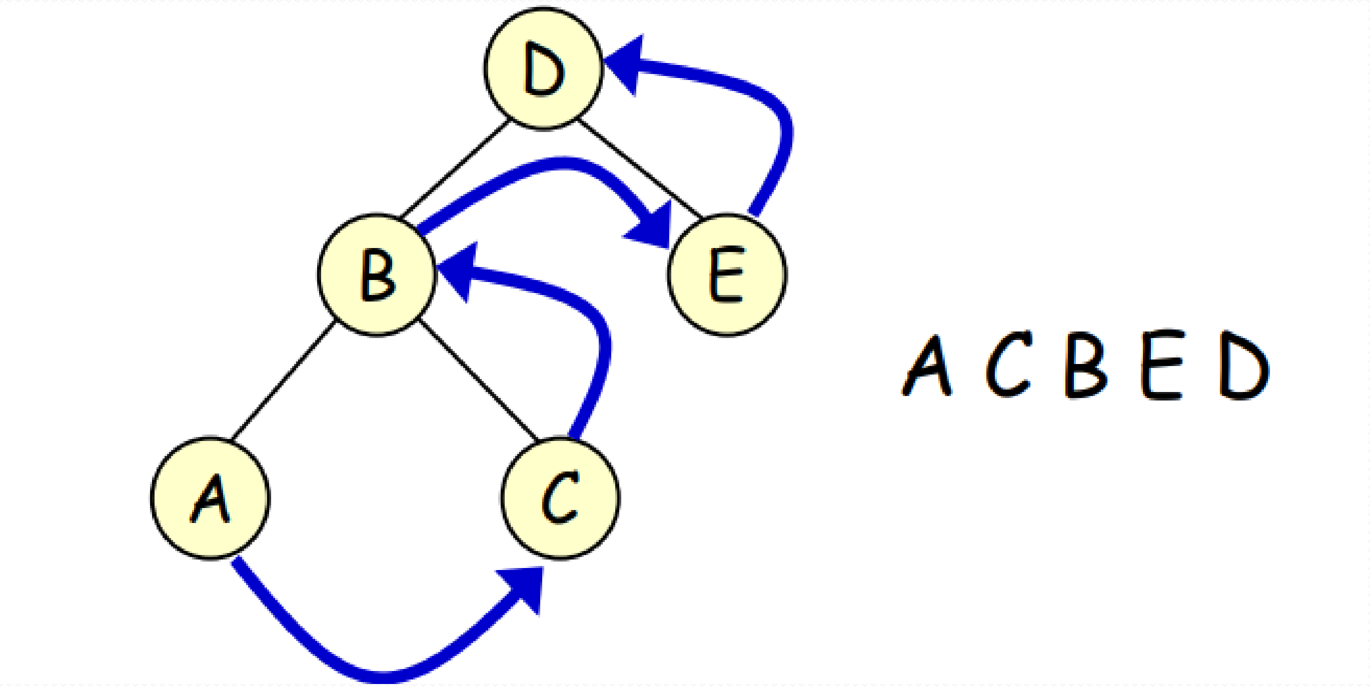
Pos Ordem: 1, 4, 7, 6, 3, 13, 14, 10, 8




```
void printPosOrder(node_t *node) {
  if (node == NULL) {
    return;
  }
  printPosOrder(node->leftChild);
  printPosOrder(node->rightChild);
  printf("%d ", node->value);
}
```

# Percurso em Pós-Ordem

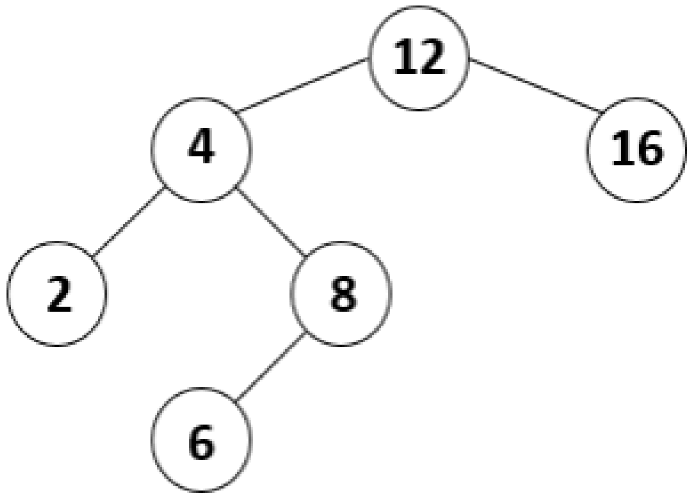
.



# Métodos de percurso em Árvore Binária

```
public void preOrdem(No a){
     a = raiz;
    if(a != null){
       System.out.print(a.dados + "   ");
       preOrdem(a.esquerda);
       preOrdem(a.direita);
    }
}
public void emOrdem(No a){
     a = raiz;
    if(a != null){
       emOrdem(a.esquerda);
       System.out.print(a.dados+"   ");
       emOrdem(a.direita);
    }
}
public void posOrdem(No a){
     a = raiz;
    if(a != null){
       posOrdem(a.esquerda);
       posOrdem(a.direita);
       System.out.print(a.dados+"   ");
    }
}
```

# Exemplo




Pré-Ordem: 12, 4, 2, 8, 6, 16

Pós-Ordem: 2, 6, 8, 4, 16, 12

# Contatos

- Email: fabio.silva321@fatec.sp.gov.br
- Linkedin: https://br.linkedin.com/in/b41a5269
- Facebook: https://www.facebook.com/fabio.silva.56211