Estruturas de Dados

Árvores Binárias de Busca

- Conceitos de Árvores Binárias
  - Métodos e algoritmos de percurso
  - Métodos e algoritmos de balanceamento
- Características Algoritmos
  - Inserção, deleção e pesquisa

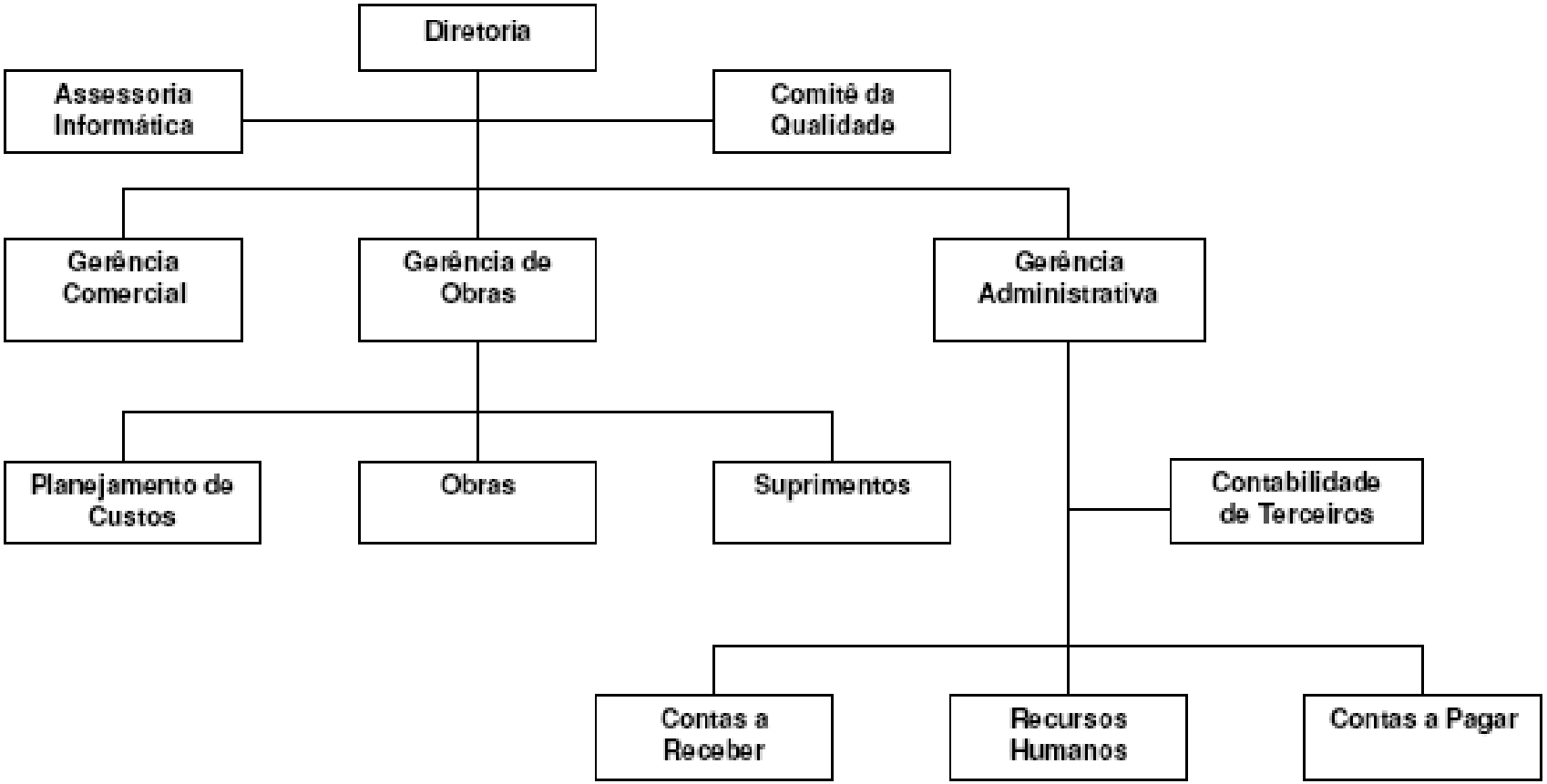
Introdução

Árvores são estruturas de dados que se caracterizam por uma organização hierárquica – **relação hierárquica** – entre seus elementos. Essa organização permite a definição de algoritmos relativamente simples, recursivos e de eficiência bastante razoável.

Introdução

- No cotidiano, diversas informações são organizadas de forma hierárquica.
- Como exemplo, podem ser citados:
  - o organograma de uma empresa;
  - a divisão de um livro em capítulos, seções, tópicos;
  - a árvore genealógica de uma pessoa.

Introdução

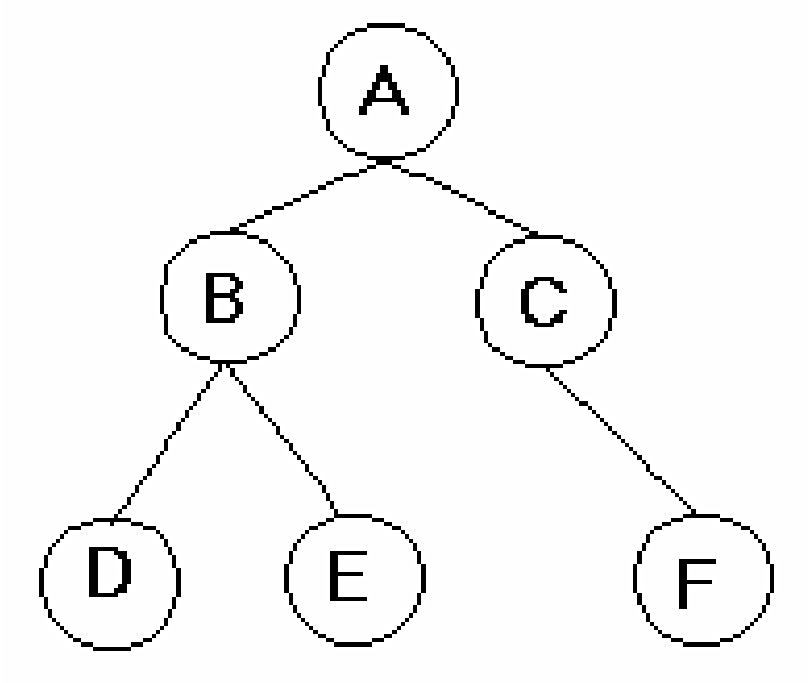

Diretoria
Assessoria Informática
Comitê da Qualidade
Gerência Comercial
Gerência de Obras
Gerência Administrativa
Planejamento de Custos
Obras
Suprimentos
Contabilidade de Terceiros
Contas a Receber
Recursos Humanos
Contas a Pagar

Introdução

- De um modo mais formal, podemos dizer que uma árvore é um conjunto finito de um ou mais ***nodos***, ***nós*** ou *vértices*, tais que:
  - existe um nodo denominado ***raiz da árvore***;
  - os demais nodos formam *n* >= 0 **conjuntos disjuntos** c_1, c_2, ..., c_n, sendo que cada um desses conjuntos também é uma árvore (denominada **subárvore**).

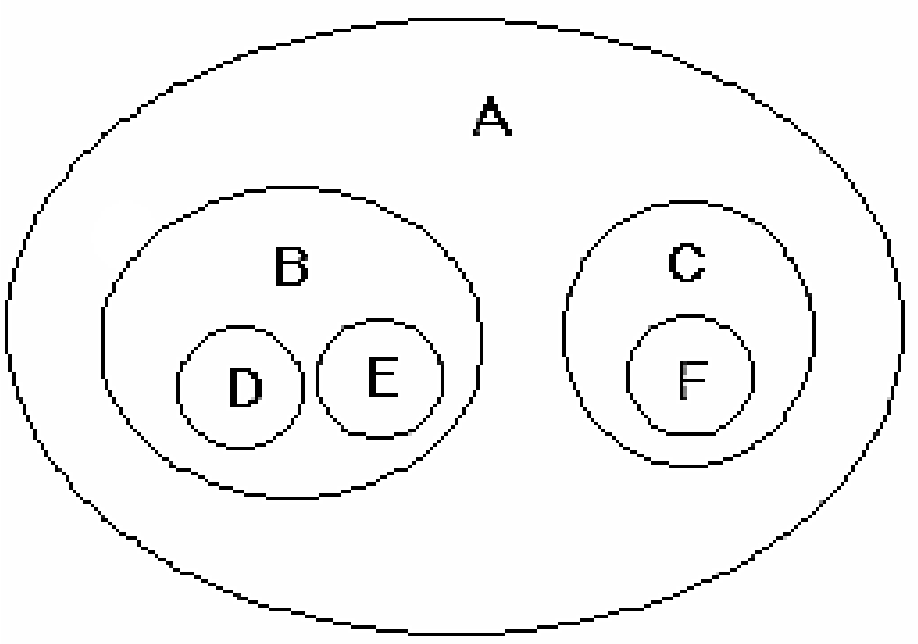
Representações

Representação hierárquica



Representações

Representação por conjuntos (diagrama de inclusão)



Representações

- **Representação por expressão parentetizada (parênteses aninhados)**
  - Cada conjunto de parênteses correspondentes contém um nodo e seus filhos. Se um nodo não tem filhos, ele é seguido por um par de parênteses sem conteúdo.

(A (B (D ( ) E ( ))) (C (F ( ))))

Representações

- **Representação por expressão não parentetizada**
  - Cada nodo é seguido por um número que indica sua quantidade de filhos, e em seguida por cada um de seus filhos, representados do mesmo modo.

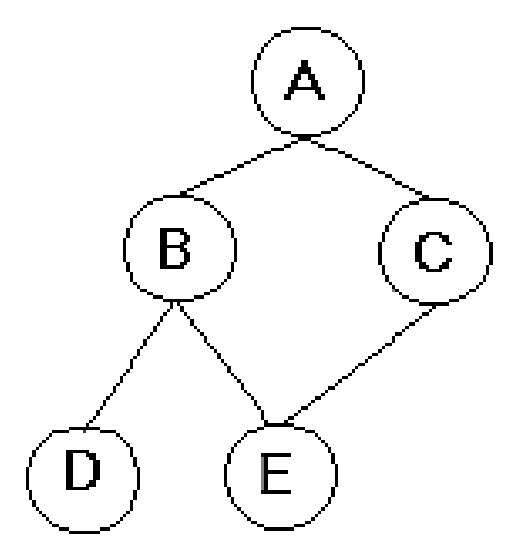
A 2 B 2 D 0 E 0 C 1 F 0

Representações

- As duas primeiras representações permitem uma melhor visualização das árvores.
- As duas últimas, por sua vez, facilitam a persistência dos nodos das árvores (em arquivos, por exemplo), possibilitando assim a sua reconstituição.

Representações

Como, por definição, os subconjuntos c_1, c_2, ..., c_n são **disjuntos**, cada nodo pode ter apenas um pai. A representação a seguir, por exemplo, não corresponde a uma árvore.



Definições

- A linha que liga dois nodos da árvore denomina-se **aresta**;
- existe um **caminho** entre dois nodos A e B da árvore, se a partir do nodo A é possível chegar ao nodo B percorrendo as arestas que ligam os nodos entre A e B;
- existe sempre um caminho entre a raiz e qualquer nodo da árvore.

Definições

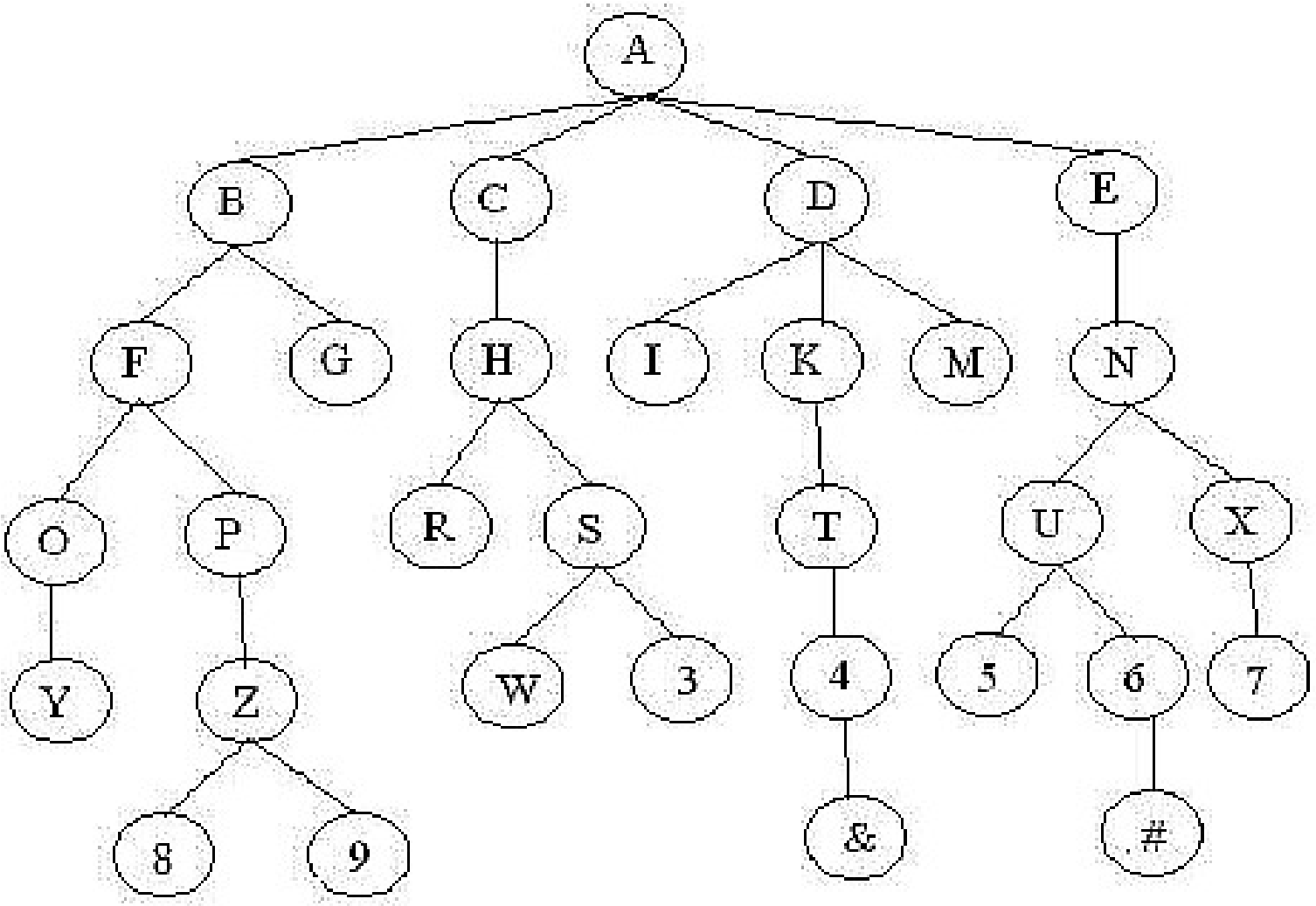
- Se houver um caminho entre A e B, começando em A diz-se que A é um **nodo ancestral** de B e B é um **nodo descendente** de A.
- Se este caminho contiver uma única aresta, diz-se que A é o **nodo pai** de B e que B é um **nodo filho** de A.
    - Dois nodos que são filhos do mesmo pai são denominados **nodos irmãos**;
    - Qualquer nodo, exceto a raiz, tem um único nodo pai.

Definições

Se um nodo não possui nodos descendentes, ele é chamado de **folha** ou **nodo terminal** da árvore;

- **grau de um nodo**: é o número de nodos filhos do mesmo. Um nodo folha tem grau zero;
- **nível de um nodo**: a raiz tem nível 0. Seus descendentes diretos têm nível 1, e assim por diante;
- **grau da árvore**: é igual ao grau do nodo de maior grau da árvore;
- **nível da árvore**: é igual ao nível do nodo de maior nível da árvore.

Exercício



Exercício

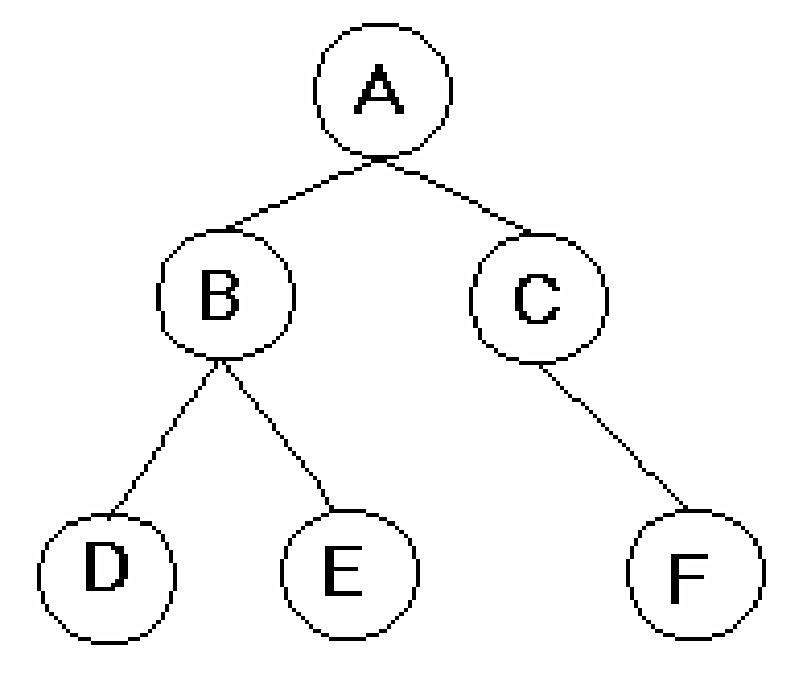
- Qual é a raiz da árvore?
- Quais são os nodos terminais?
- Qual o grau da árvore?
- Qual o nível da árvore?
- Quais são os nodos descendentes do nodo **D**?
- Quais são os nodos ancestrais do nodo **#**?
- Os nodos **4** e **5** são nodos irmãos?
- Há caminho entre os nodos **C** e **S**?
- Qual o nível do nodo **5**?
- Qual o grau do nodo **A**?

Árvores Binárias

- A inclusão de limitações estruturais define tipos específicos de árvores.
- Até agora, as árvores vistas não possuíam nenhuma limitação quanto ao **grau máximo** de cada nodo.
- Uma ***árvore binária*** é uma árvore cujo grau máximo de cada nodo é 2. Essa limitação define uma nomenclatura específica:
  - os filhos de um nodo são classificados de acordo com sua posição relativa à raiz;
  - assim, distinguem-se o filho da esquerda e o filho da direita e, consequentemente, a **subárvore da esquerda** e a **subárvore da direita**.

Árvores Binárias

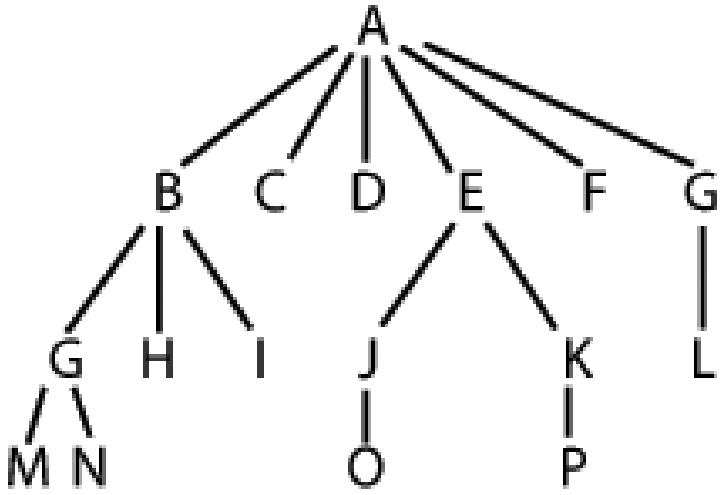
Exemplo de árvore binária

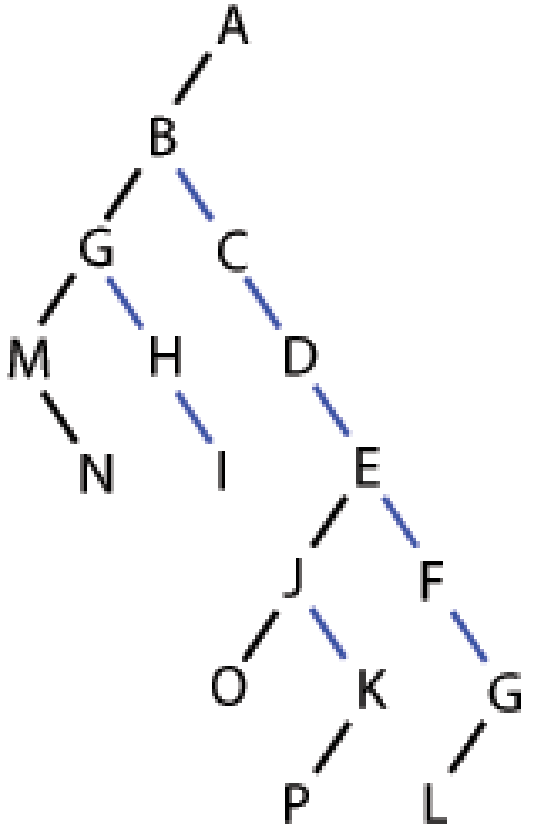


Transformações

- É possível transformar uma árvore ***n-ária*** em uma árvore binária através dos seguintes passos:
  - a raiz da árvore (subárvore) será a raiz da árvore (subárvore) binária;
  - O nodo filho mais à esquerda da raiz da árvore (subárvore) será o nodo filho à esquerda da raiz da árvore (subárvore) binária.
  - Cada nodo irmão de A, da esquerda para a direita, será o nodo filho à direita do nodo irmão da esquerda, até que todos os nodos filhos da raiz da árvore (subárvore) já tenham sido incluídos na árvore binária em construção.

Transformações





Modelagem: nodo de uma árvore binária

Necessitamos:

- um ponteiro para o filho localizado à esquerda;
- um ponteiro para o filho localizado à direita;
- um ponteiro para a informação que vamos armazenar.

Pseudo-código:

```
classe tNodo {
   tNodo *filhoEsquerda;
   tNodo *filhoDireita;
   TipoInfo *info;
 };
```

Construção de uma árvore binária

- Árvores como estruturas para organizar informações:
  - dados a serem inseridos em uma árvore são dados ordenáveis de alguma forma. Exemplo mais simples: números inteiros.
- A árvore deverá possuir altura mínima:
  - caminhos médios de busca mínimos para uma mesma quantidade de dados.
- Como fazer isso?
  - garantir profundidades médias mínimas, preencher ao máximo cada nível antes de partir para o próximo e distribuir homogeneamente os nodos para a esquerda e direita.

Construção de uma árvore binária

- **Algoritmo:**
  - use um nodo para a raiz;
  - gere a subárvore esquerda com **nodosÀEsquerda = númeroDeNodos / 2** nodos, usando este mesmo procedimento;
  - gere a subárvore direita com **nodosÀDireita = númeroDeNodos – nodosÀEsquerda - 1** nodos, usando este mesmo procedimento.

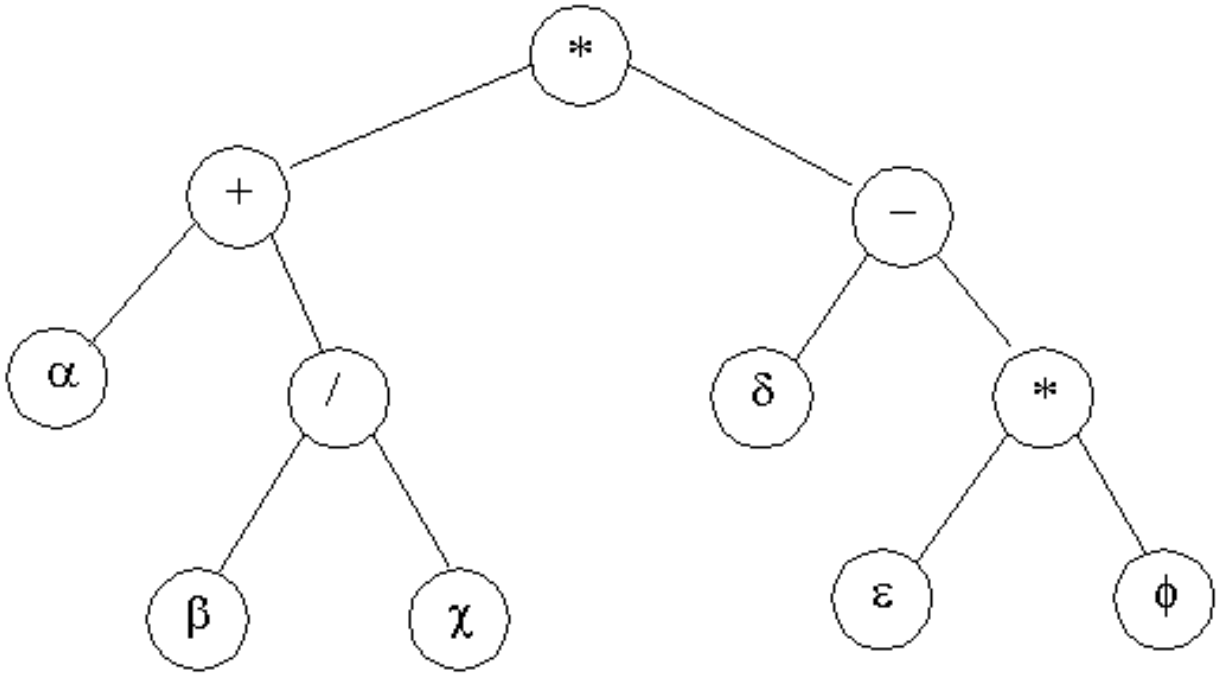
Árvore binária balanceada

```
tNodo* FUNÇÃO constróiÁrvore(inteiro númeroDeNodos)
   inteiro nodosÀEsquerda, nodosÀDireita;
   TipoInfo *info;
   tNodo *novoNodo;
   início
    se númeroDeNodos = 0 então
        retorna NULO;
    nodosÀEsquerda <- númeroDeNodos / 2;
    nodosÀDireita <- númeroDeNodos – nodosÀEsquerda – 1;
    aloque(info);
    ler(info);
    aloque(novoNodo);
    novoNodo->info <- info;
    novoNodo->filhoEsquerda <- constróiÁrvore(nodosÀEsquerda);
    novoNodo->filhoDireita <- constróiÁrvore(nodosÀDireita);
    retorna novoNodo;
   fim
```

Percurso em árvores binárias

- O percurso em árvores binárias corresponde ao caminhamento executado em listas:
  - partimos de um nodo inicial (**raiz**) e visitamos todos os demais nodos em uma ordem previamente especificada.
- Como exemplo, considere uma árvore binária utilizada para representar uma expressão (com as seguintes restrições):
  - cada operador representa uma bifurcação;
  - seus dois operandos correspondentes são representados por suas subárvores.

Percurso em árvores binárias



Representação da expressão $(\alpha+\beta/\chi)*(\delta-\varepsilon*\phi)$ como árvore

Percurso em árvores binárias

- Existem três ordens para se percorrer uma árvore binária que são consequência natural da estrutura da árvore:
  - **Preordem(r,e,d)** – *Preorder*
  - **Emordem(e,r,d)** – *Inorder*
  - **Pósordem(e,d,r)** – *Postorder*

Percurso em árvores binárias

- Essas ordens são definidas recursivamente (definição natural para uma árvore) e em função da **raiz(r)**, da **subárvore esquerda(e)** e da **subárvore direita(d)**:
  - Preordem(r,e,d): visite a raiz **ANTES** das subárvores
  - Emordem(e,r,d): visite primeiro a subárvore **ESQUERDA**, depois a **RAIZ** e depois a subárvore **DIREITA**
  - Pósordem(e,d,r): visite a raiz **DEPOIS** das subárvores
- As subárvores são **SEMPRE** visitadas da esquerda para a direita

Percurso em árvores binárias

- Se percorrermos a árvore anterior usando as ordens definidas, teremos as seguintes seqüências:
  - Preordem: **`* + a / b c - d * e f`** (notação prefixada)
  - Emordem: **`a + b / c * d - e * f`** (notação infixada)
  - Pósordem: **`a b c / + d e f * - *`** (notação posfixada)

Percurso em Preordem

```
FUNÇÃO Preordem(tNodo *raiz)
  início
   se raiz != NULO então
       imprime(raiz->info);
       Preordem(raiz->filhoEsquerda);
       Preordem(raiz->filhoDireita);
   fim se
  fim
```

Percurso Emordem

```
FUNÇÃO Emordem(tNodo *raiz)
   início
   se raiz != NULO então
      Emordem(raiz->filhoEsquerda);
      imprime(raiz->info);
      Emordem(raiz->filhoDireita);
   fim se
   fim
```

Percurso em Pósordem

```
FUNÇÃO Pósordem(tNodo *raiz)
  início
  se raiz != NULO então
    Pósordem(raiz->filhoEsquerda);
    Pósordem(raiz->filhoDireita);
    imprime(raiz->info);
  fim se
  fim
```

Árvores Binárias de Busca

- **Árvores (binárias)** são muito utilizadas para se representar um grande conjunto de dados onde se deseja encontrar um elemento de acordo com a sua **chave**.
- **Definição - Árvore Binária de Busca (Niklaus Wirth)**:
  - "*Uma árvore que se encontra organizada de tal forma que, para cada nodo t_i, todas as chaves (info) da subárvore à esquerda de t_i são menores que (ou iguais a) t_i e à direita são maiores que t_i*".
- Termo em Inglês: ***Search Tree***.

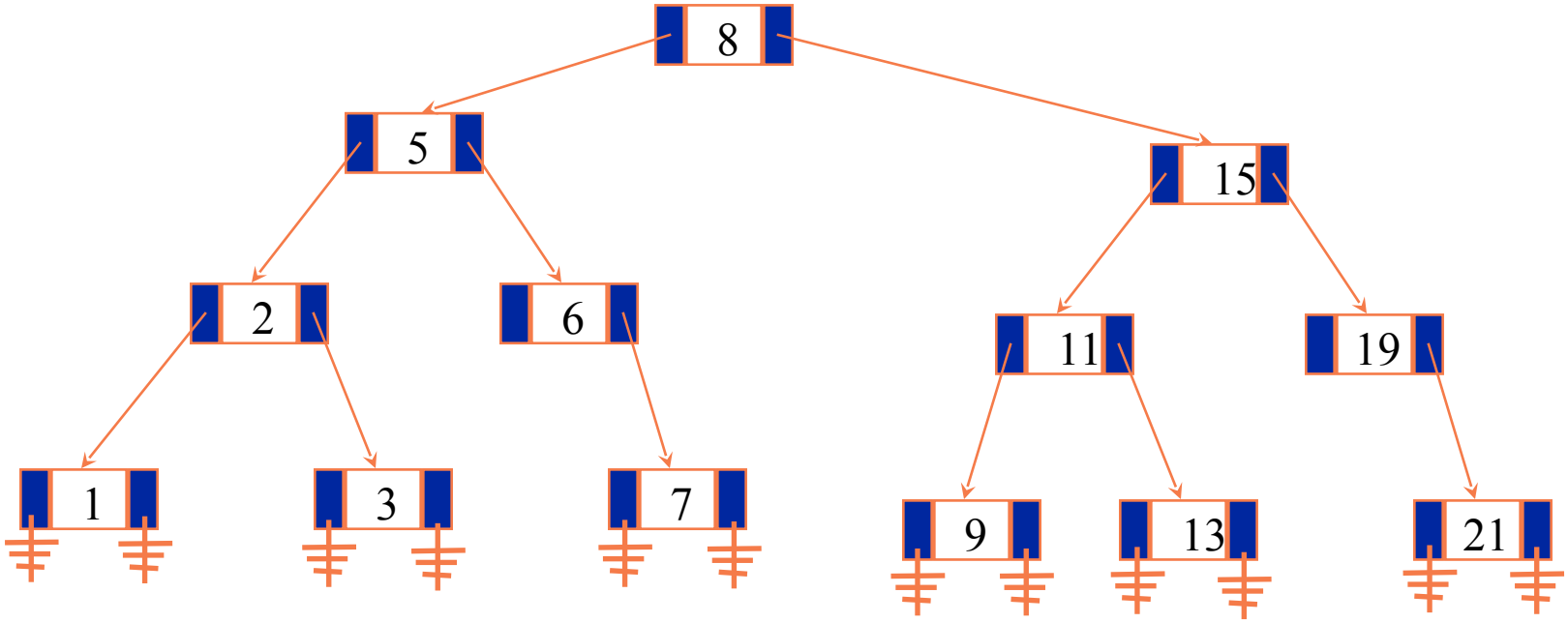
Características

- Em uma árvore binária de busca é possível encontrar-se qualquer chave existente descendo-se pela árvore:
  - sempre à **esquerda** toda vez que a chave procurada for **menor** do que a chave do nodo visitado;
  - sempre à **direita** toda vez que for **maior**.

Características

- A escolha da direção de busca só depende da chave que se procura e da chave que o nodo atual possui.
- A busca de um elemento em uma árvore balanceada com *n* elementos toma **tempo médio < log(*n*)**, sendo a busca então O(log *n*).
- Graças à estrutura de **árvore** a busca poderá ser feita com apenas **log(*n*)** comparações de elementos.

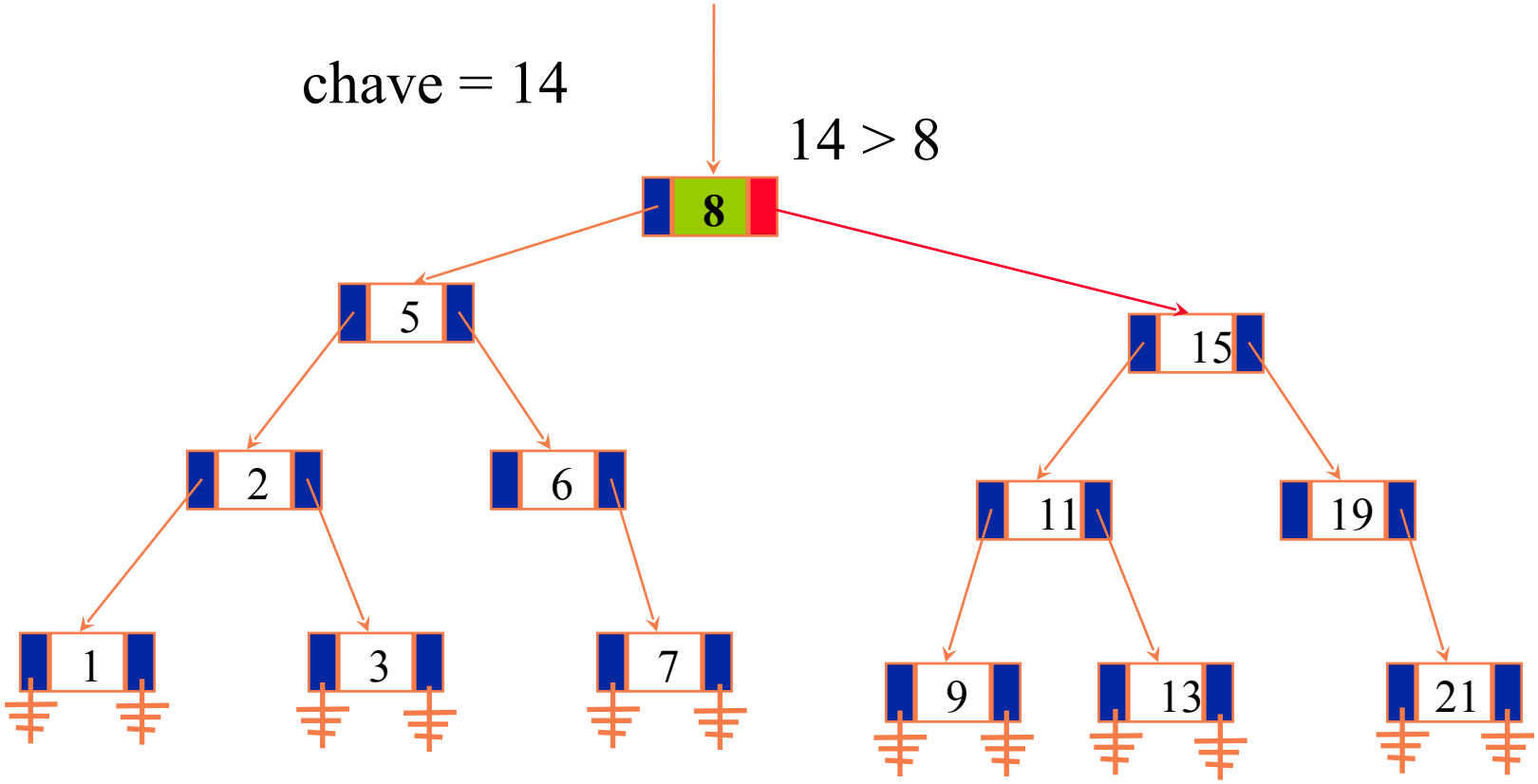
Exemplo de árvore binária de busca



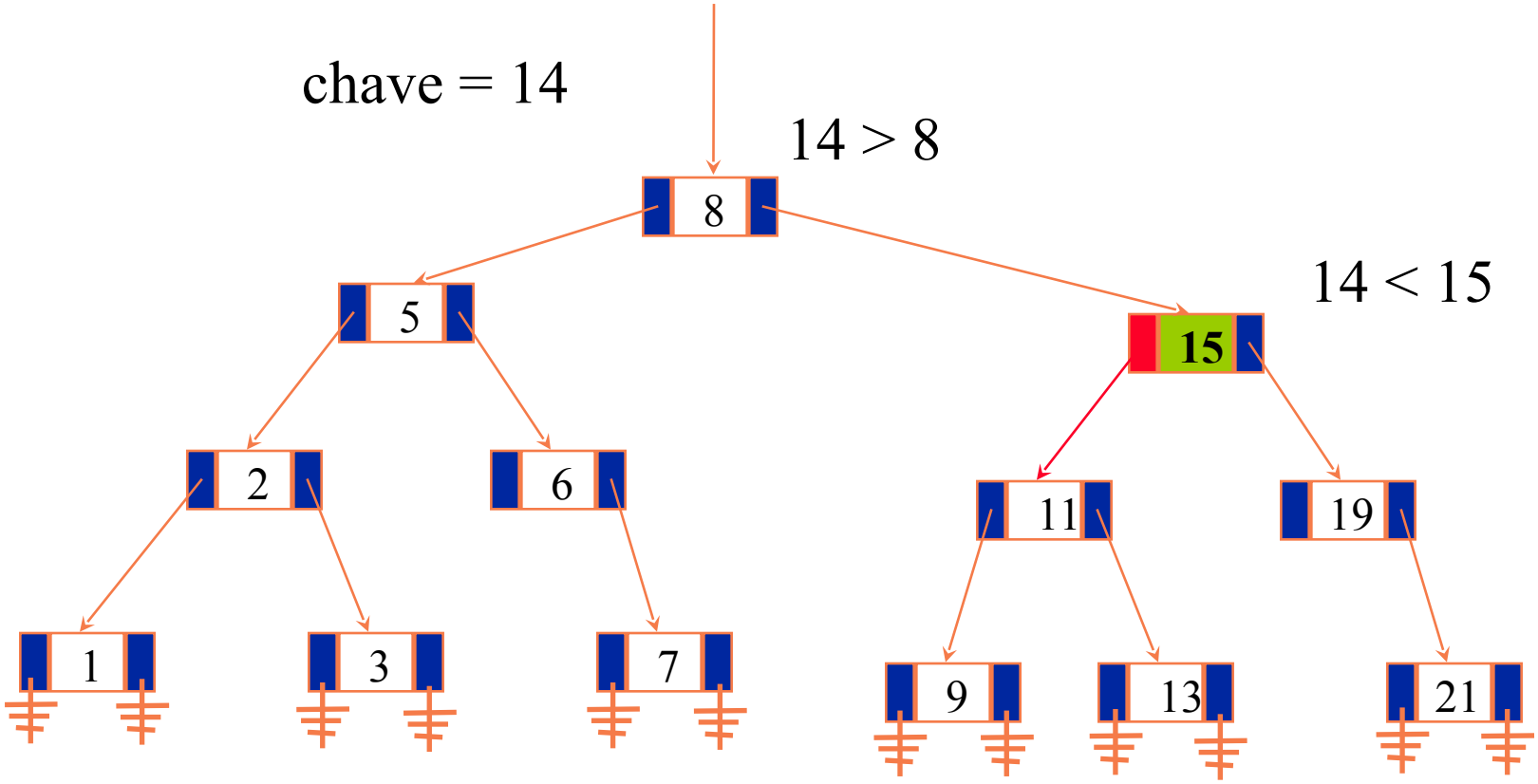
Algoritmo de busca

```
tNodo* FUNÇÃO busca (chave: tInfo, ptr: *tNodo)
  início
     enquanto (ptr ~= NULO
           E ptr->info ~= chave) faça
           // Esquerda ou direita.
           se (ptr->info < chave) então
                ptr <- ptr->filhoÀDireita
           senão
                ptr <- ptr->filhoÀEsquerda;
           fim se
     fim enquanto
     retorne ptr;
  fim
```

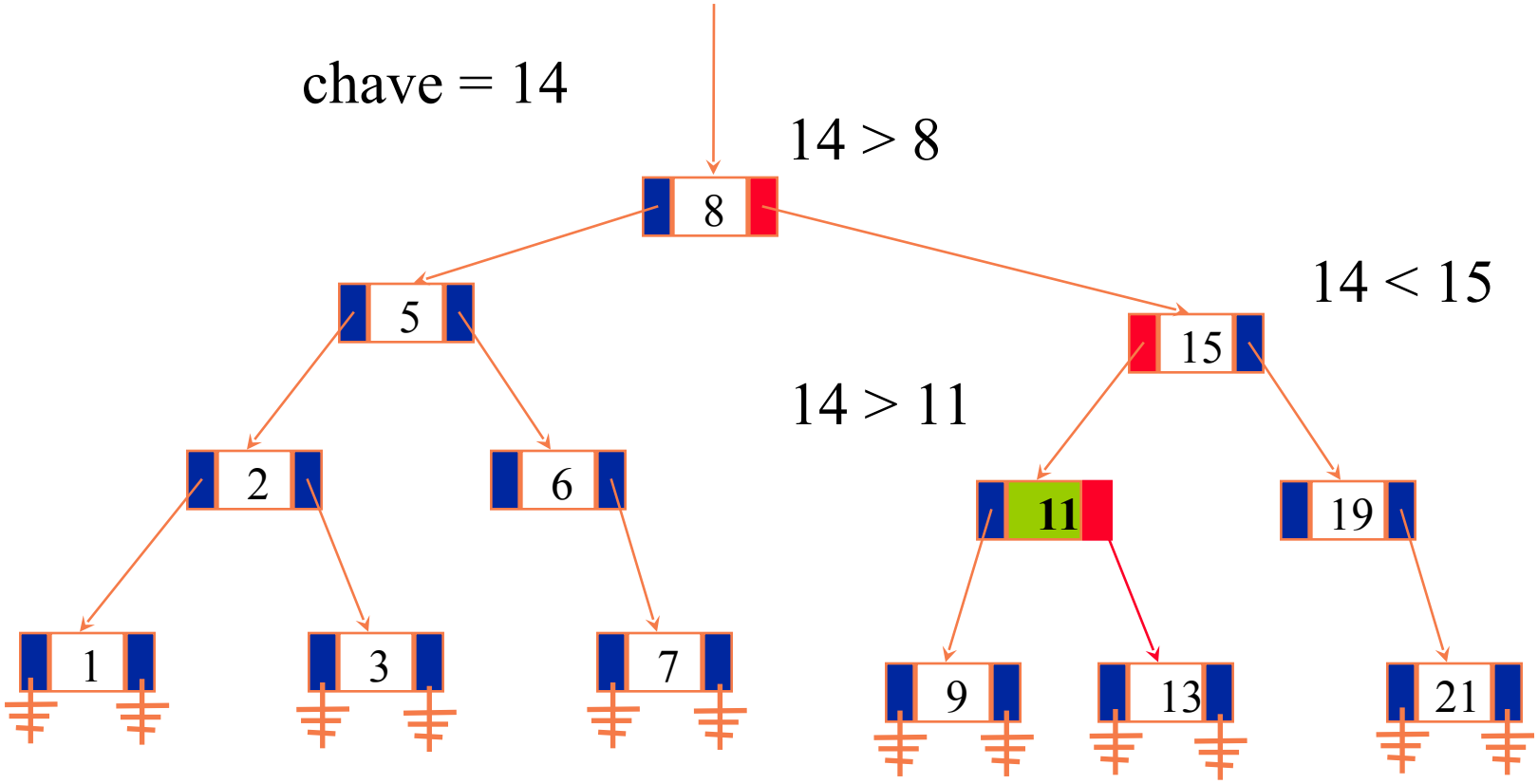
Exemplo: inserção de um elemento com chave = 14



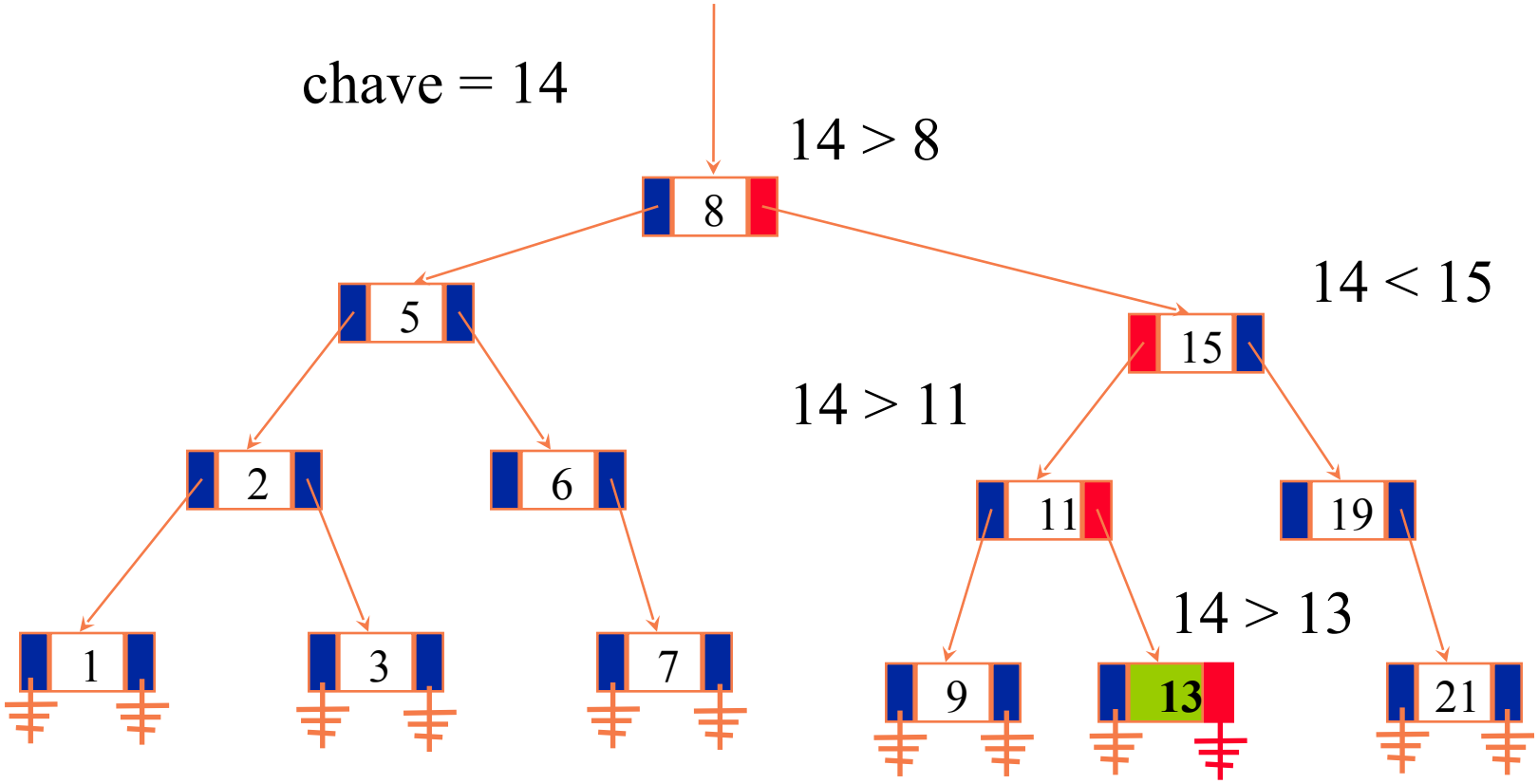
Exemplo: inserção de um elemento com chave = 14



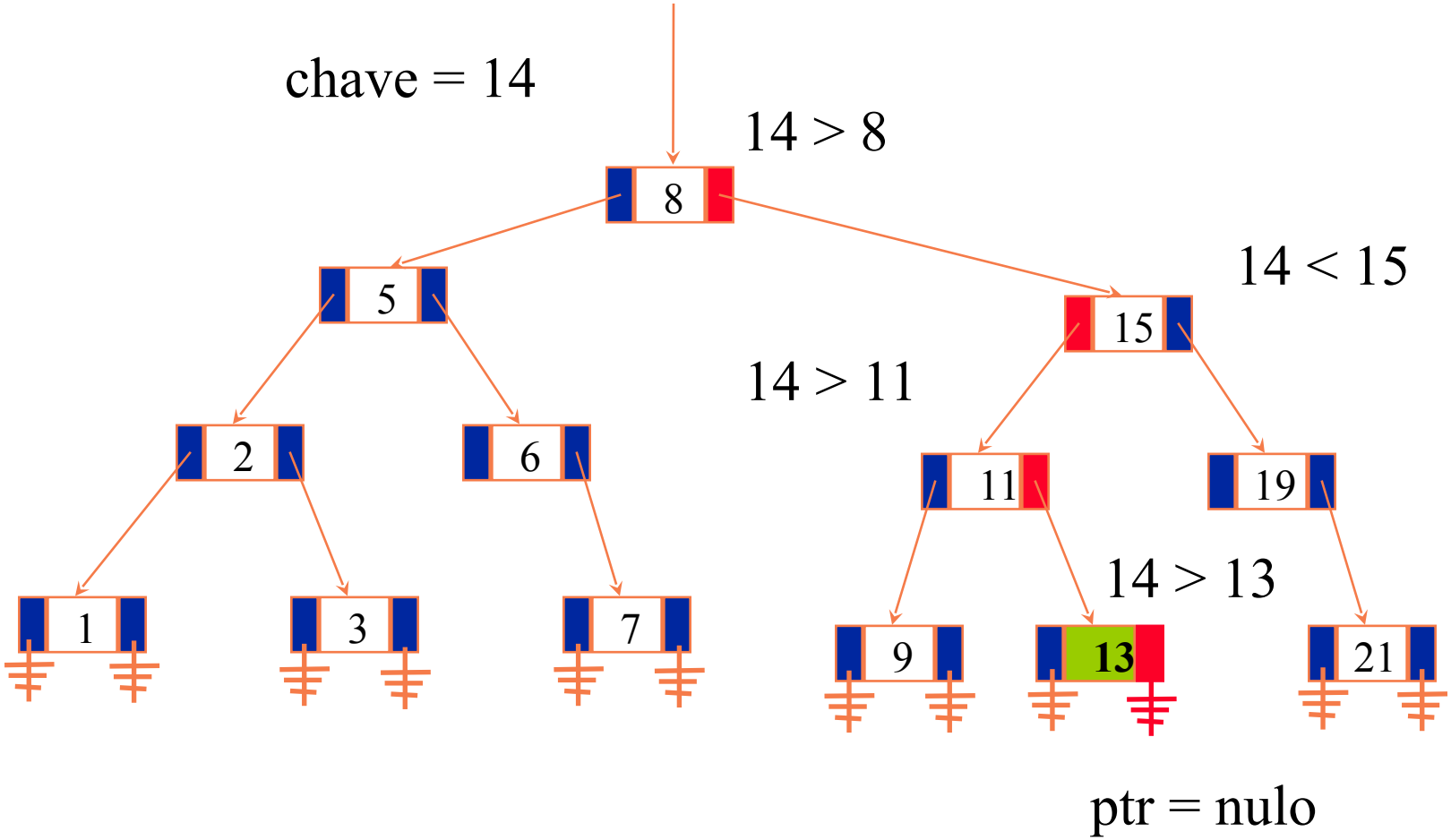
Exemplo: inserção de um elemento com chave = 14



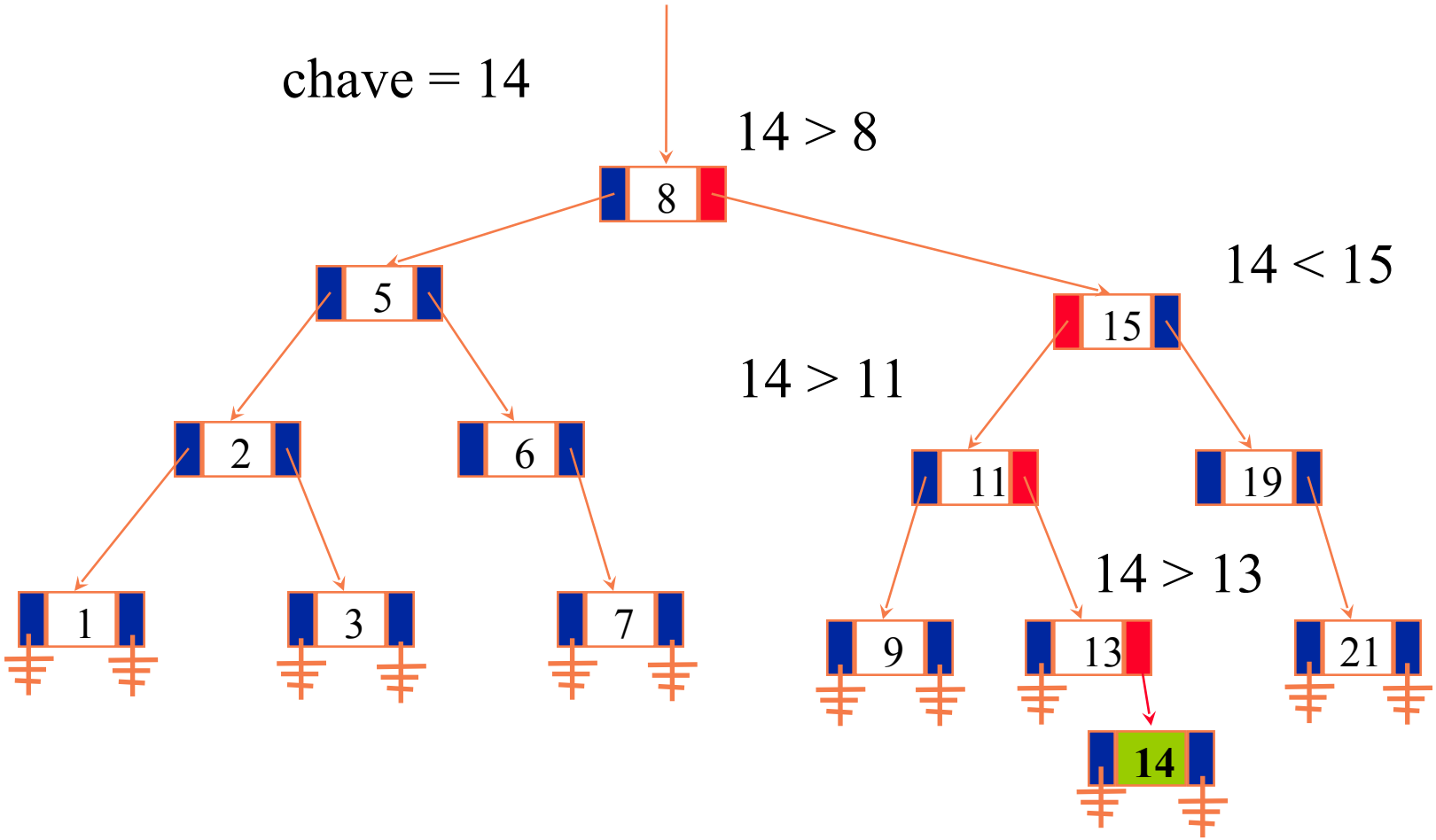
Exemplo: inserção de um elemento com chave = 14



Exemplo: inserção de um elemento com chave = 14



Exemplo: inserção de um elemento com chave = 14



Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

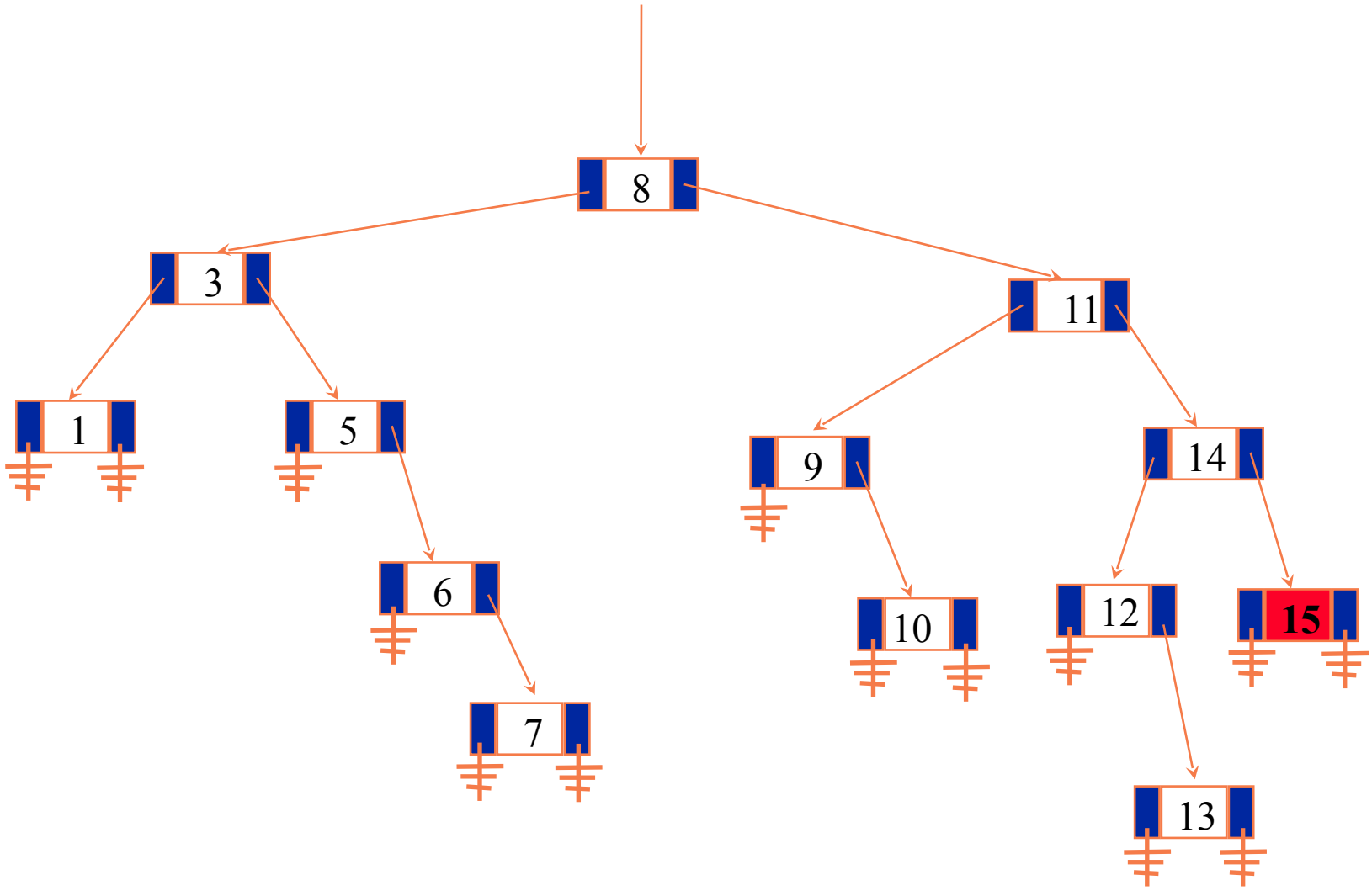
Algoritmo de inserção

```
tNodo* FUNÇÃO inserção (raiz: *tNodo, info: tInfo)
  oNovo: *tNodo;
  início
    se (info < raiz->info) então
      // Inserção à esquerda.
      se (raiz->filhoÀEsquerda = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀEsquerda <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀEsquerda, info);
      fim se
    senão
      // Inserção à direita.
      se (raiz->filhoÀDireita = NULO) então
        oNovo <- aloque(tNodo);
        oNovo->info <- info;
        oNovo->filhoÀEsquerda <- NULO;
        oNovo->filhoÀDireita <- NULO;
        raiz->filhoÀDireita <- oNovo;
      senão
        raiz <- inserção(raiz->filhoÀDireita, info);
      fim se
    fim se
  fim
```

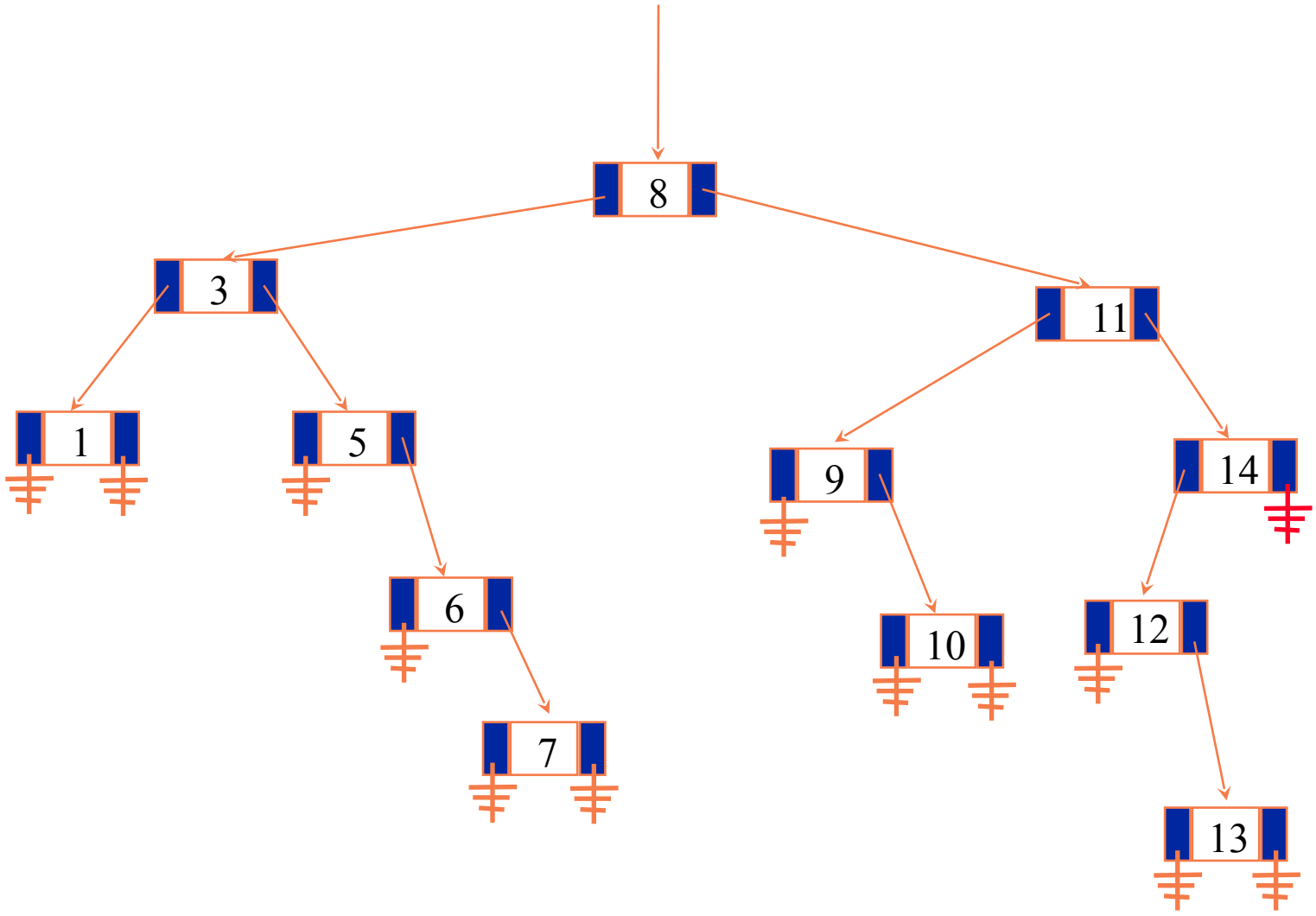
Algoritmo de deleção

- A deleção é mais complexa do que a inserção.
- A razão básica é que a **característica** organizacional da árvore não deve ser quebrada:
  - a subárvore da direita de um nodo não deve possuir **chaves menores** do que o pai do nodo eliminado;
  - a subárvore da esquerda de um nodo não deve possuir **chaves maiores** do que o pai do nodo eliminado.
- Para garantir isso, o algoritmo de deleção deve **remanejar** os nodos.

Exemplo: deleção do nodo com chave = 15



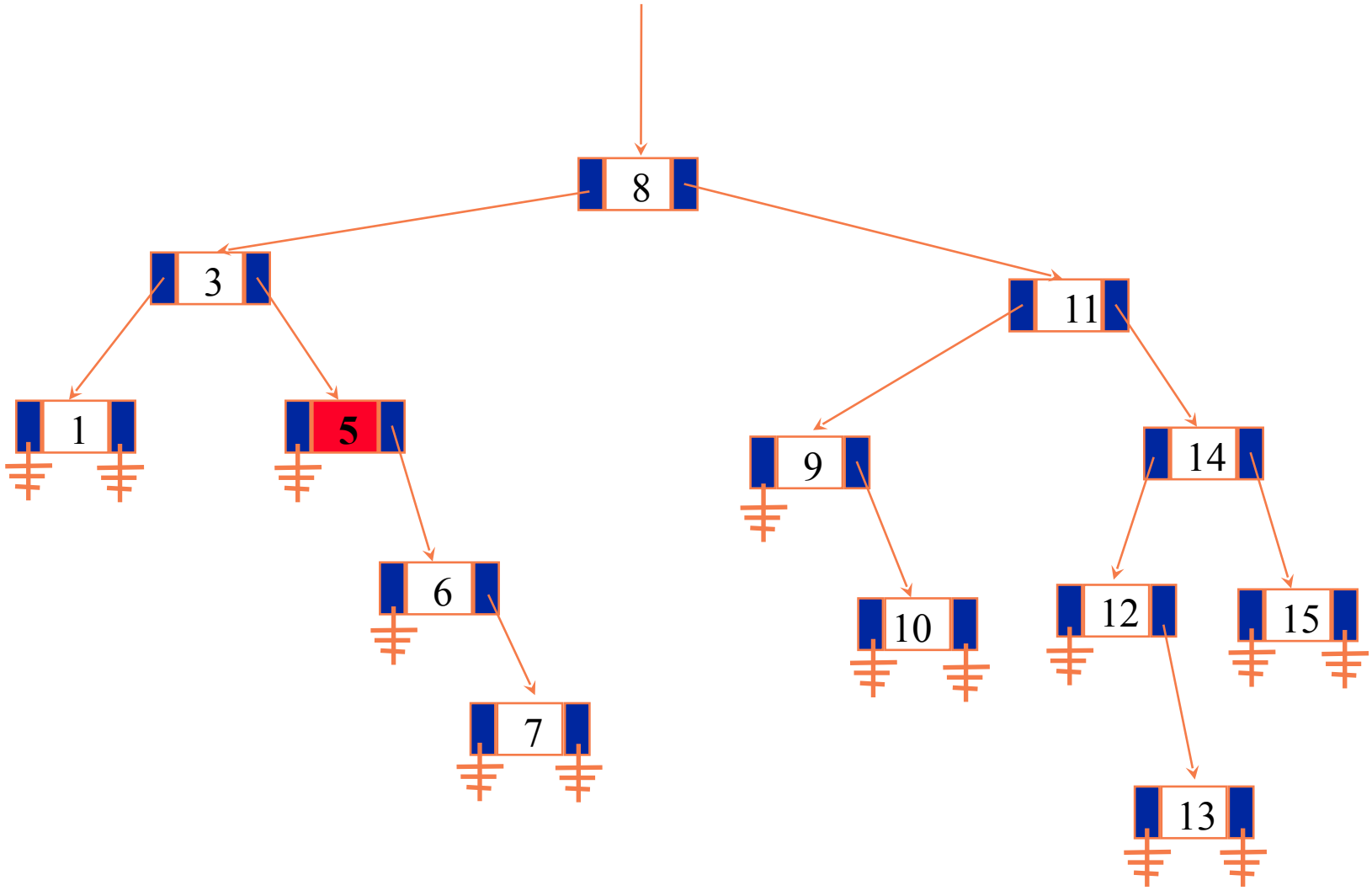
Exemplo: deleção do nodo com chave = 15



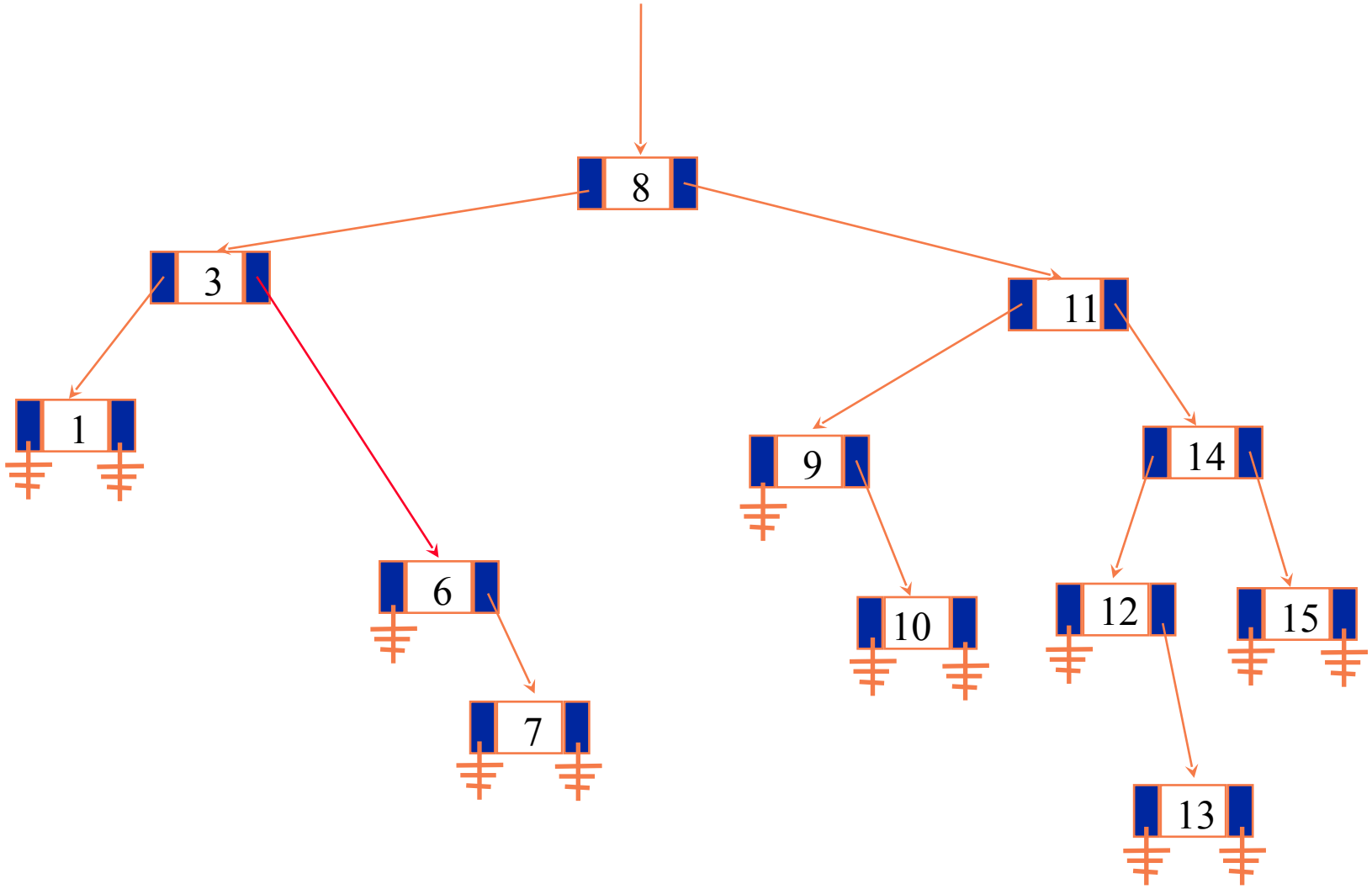
Deleção em uma árvore de busca binária

- Se o nodo possuir somente uma subárvore filha:
  - podemos simplesmente **mover** esta subárvore toda para cima;
  - o único sucessor do nodo a ser excluído será um dos **sucessores diretos** do pai do nodo a ser eliminado;
  - se o nodo a ser excluído é filho esquerdo de seu pai, o seu filho será o **novo filho esquerdo** deste e vice-versa.

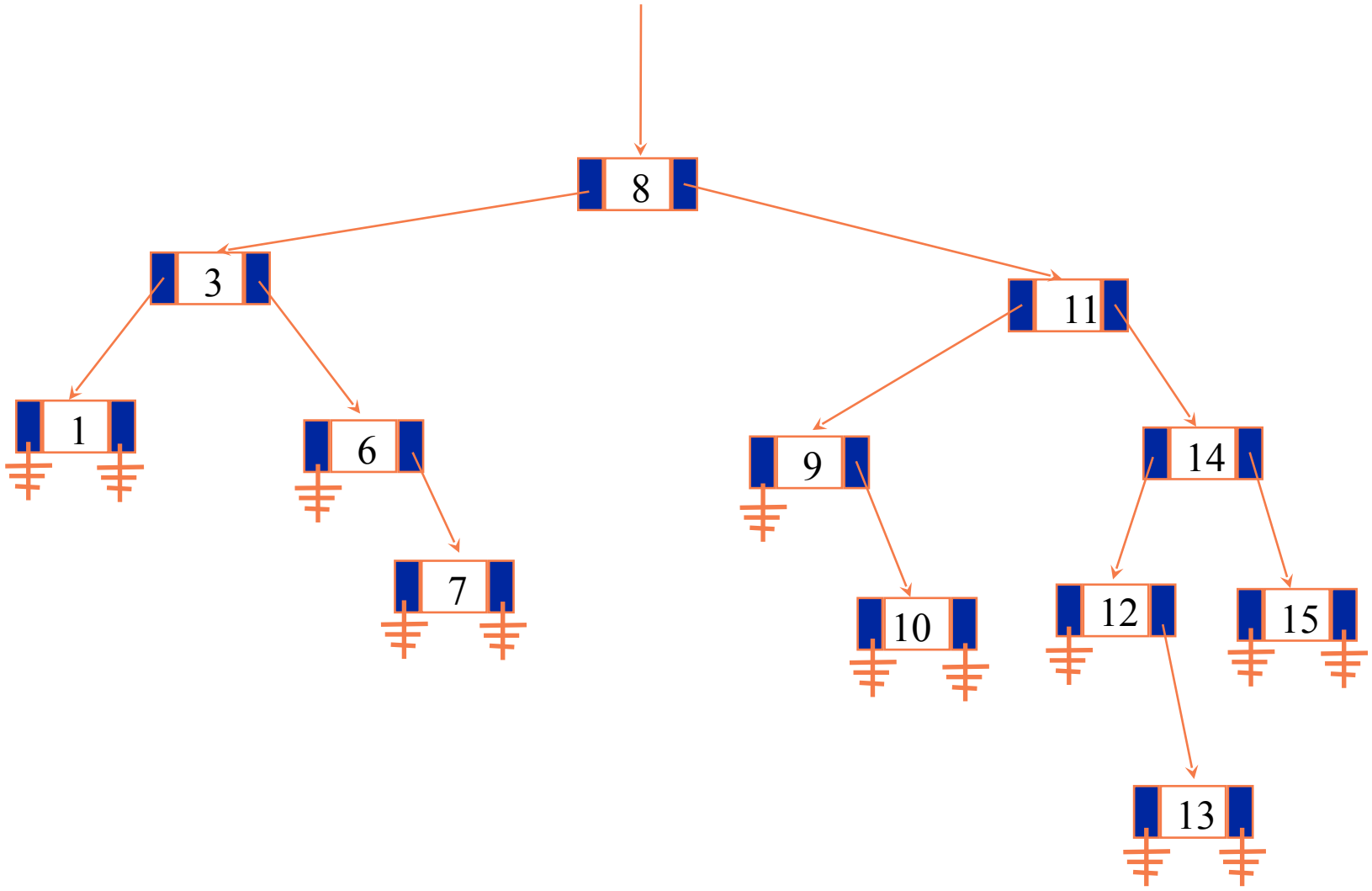
Exemplo: deleção do nodo com chave = 5



Exemplo: deleção do nodo com chave = 5



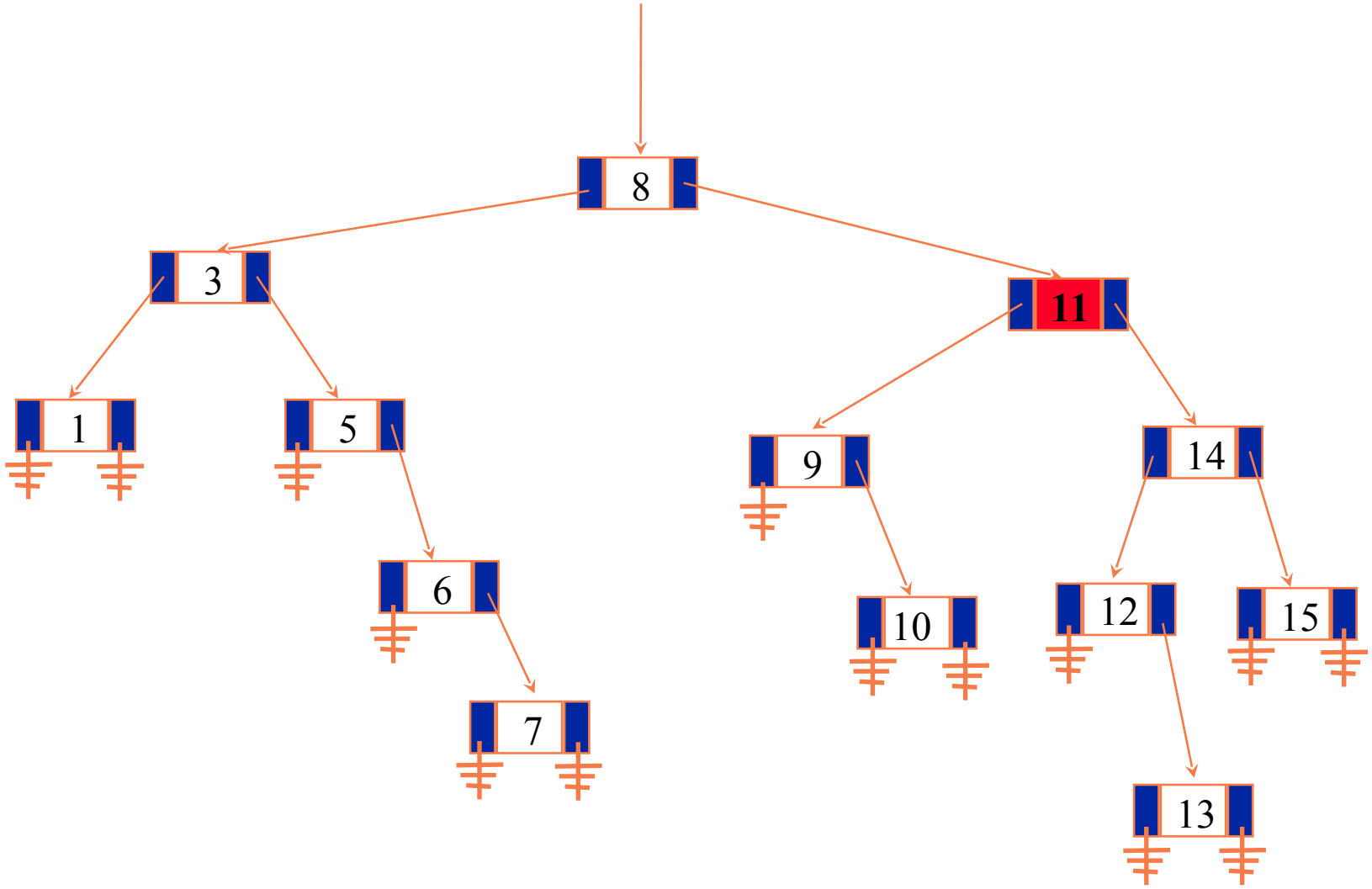
Exemplo: deleção do nodo com chave = 5



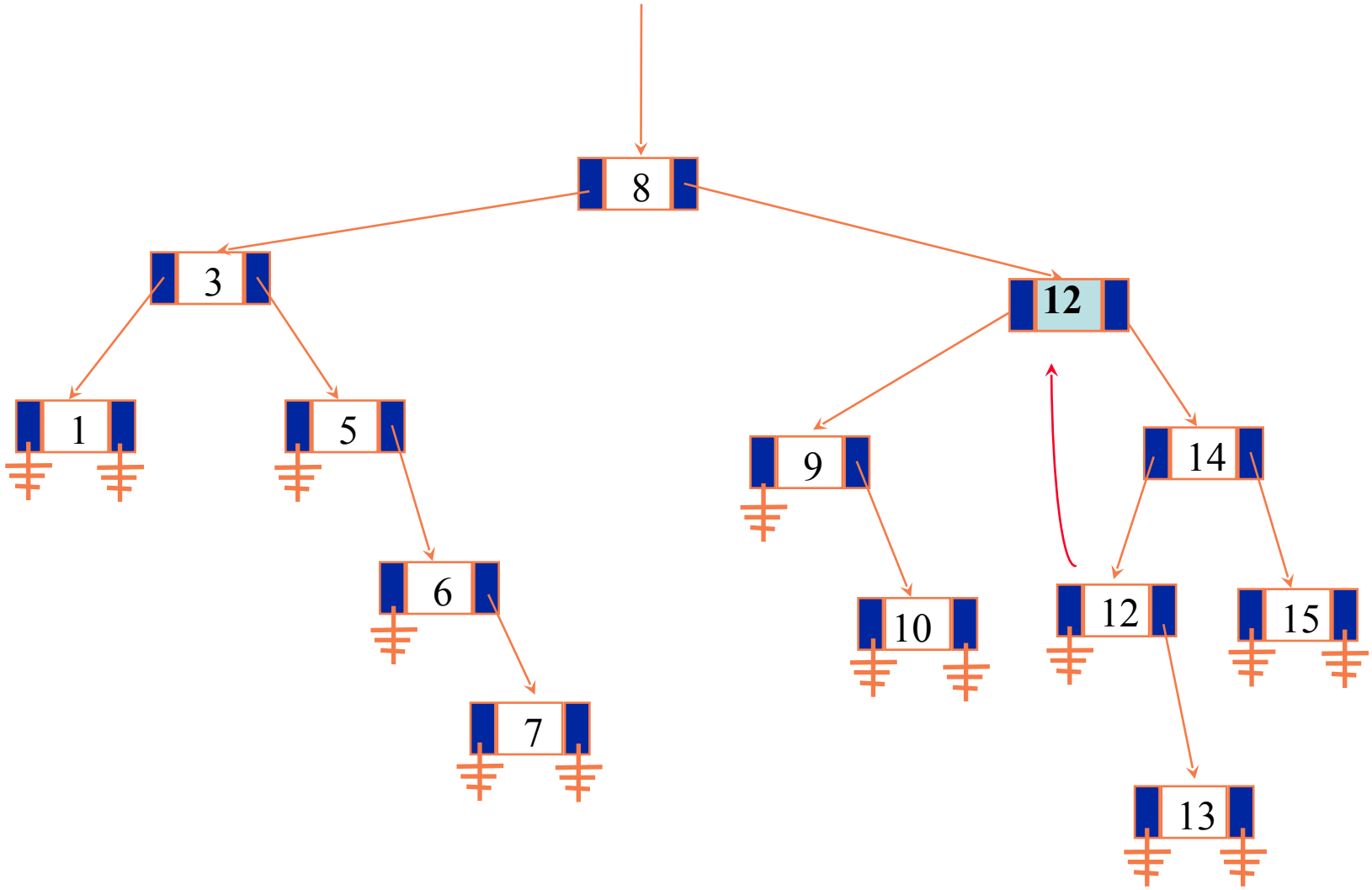
Deleção em uma árvore de busca binária

- Se o nodo possuir duas subárvores filhas:
  - se o filho à direita não possui **subárvore esquerda**, é ele quem ocupa o seu lugar;
  - se possuir uma subárvore esquerda, a **raiz desta será movida para cima** e assim por diante;
  - a estratégia geral (Mark Allen Weiss) é sempre **substituir a chave retirada pela menor chave da subárvore direita**.

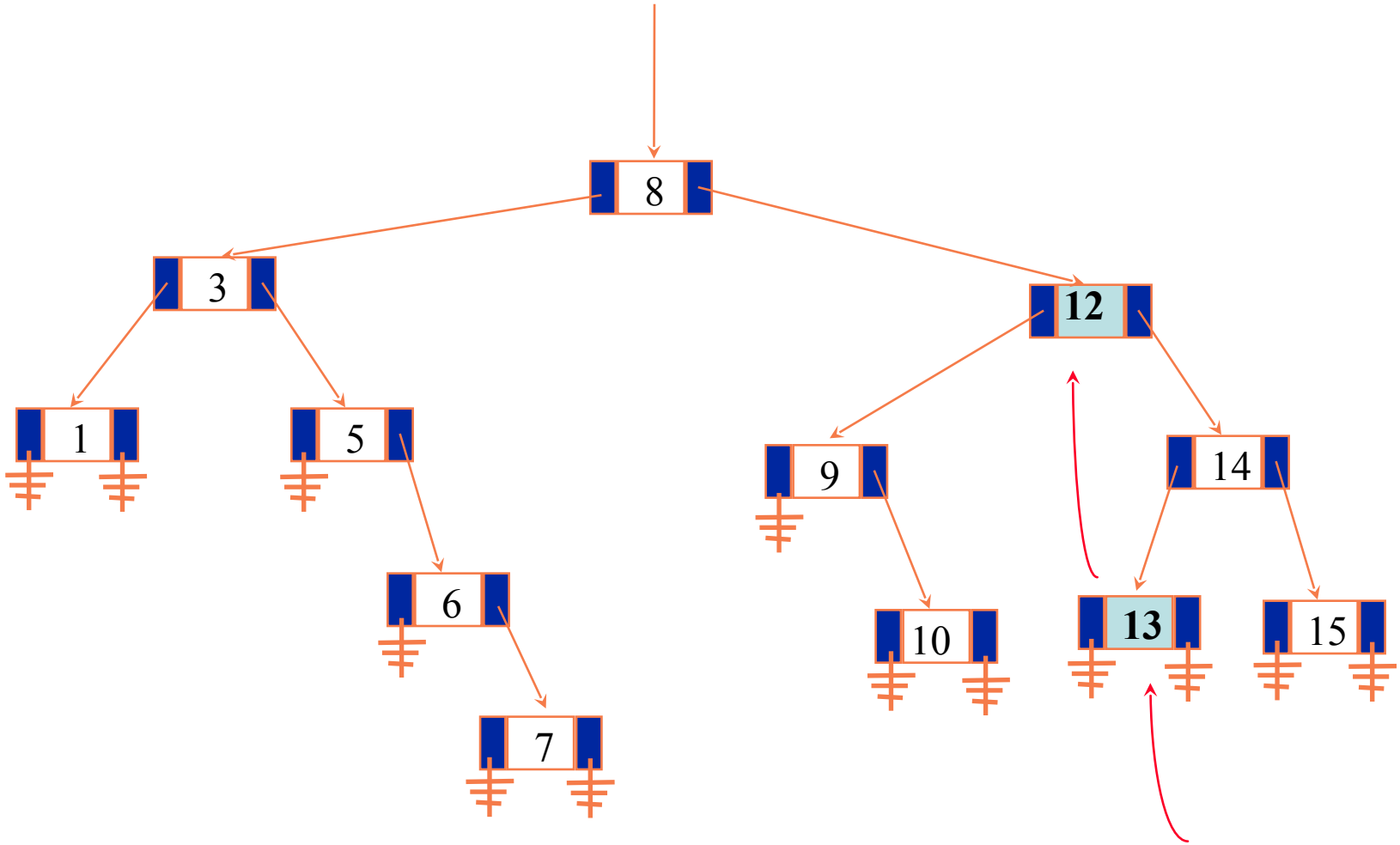
Exemplo: deleção do nodo com chave = 11



Exemplo: deleção do nodo com chave = 11



Exemplo: deleção do nodo com chave = 11



Algoritmo de deleção

```
tNodo* FUNÇÃO delete(info: tInfo, arv: *tNodo)
  tmp, filho: *tNodo;
  início
    se (arv = NULO) então
      retorne arv
    senão
      se (info < arv->info) // Vá à esquerda.
        arv->filhoÀEsquerda <- delete(info, arv->filhoÀEsquerda);
        retorne arv;
      senão
        se (info > arv->info) // Vá à direita.
          arv->filhoÀDireita <- delete(info, arv->filhoÀDireita);
          retorne arv;
        senão // Encontrei elemento que quero deletar.
        (CONTINUA)
```

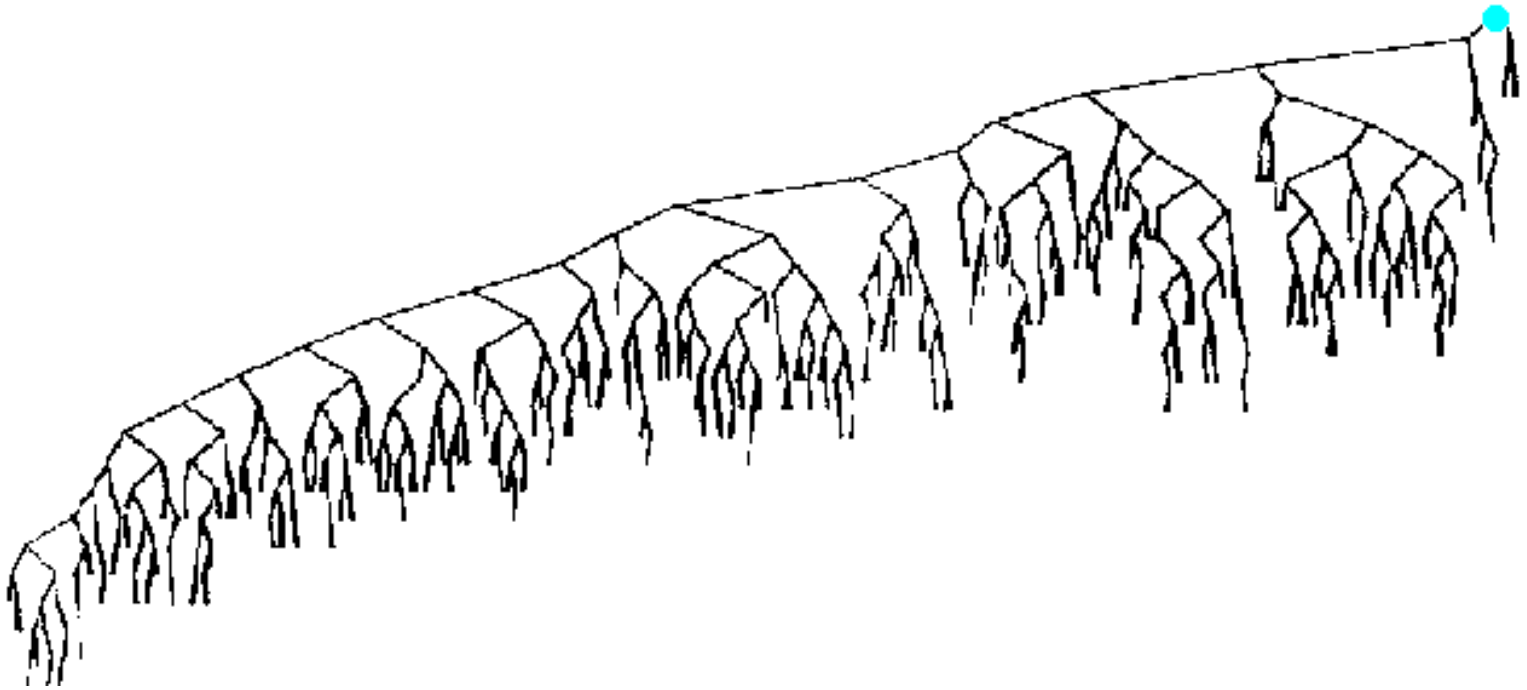
Algoritmo de deleção

```
        (CONTINUAÇÃO)
        se (arv->filhoÀDireita ~= NULO E arv->filhoÀEsquerda ~= NULO) // 2 filhos.
          tmp <- mínimo(arv->filhoÀDireita);
          arv->info <- tmp->info;
          arv->filhoÀDireita <- delete(arv->info, arv->filhoÀDireita);
          retorne arv;
        senão // 1 filho.
          tmp <- arv;
          se (arv->filhoÀDireita ~= NULO) então // Filho à direita.
            filho <- arv->filhoÀDireita;
            retorne filho;
          senão
            se (arv->filhoÀEsquerda ~= NULO) então // Filho à esquerda.
              filho <- arv->filhoÀEsquerda;
              retorne filho;
            senão // Folha.
              libere arv;
              retorne NULO;
            fim se
          fim se
        fim se
      fim se
    fim se
  fim se
fim
```

Problemas com árvores binárias de busca

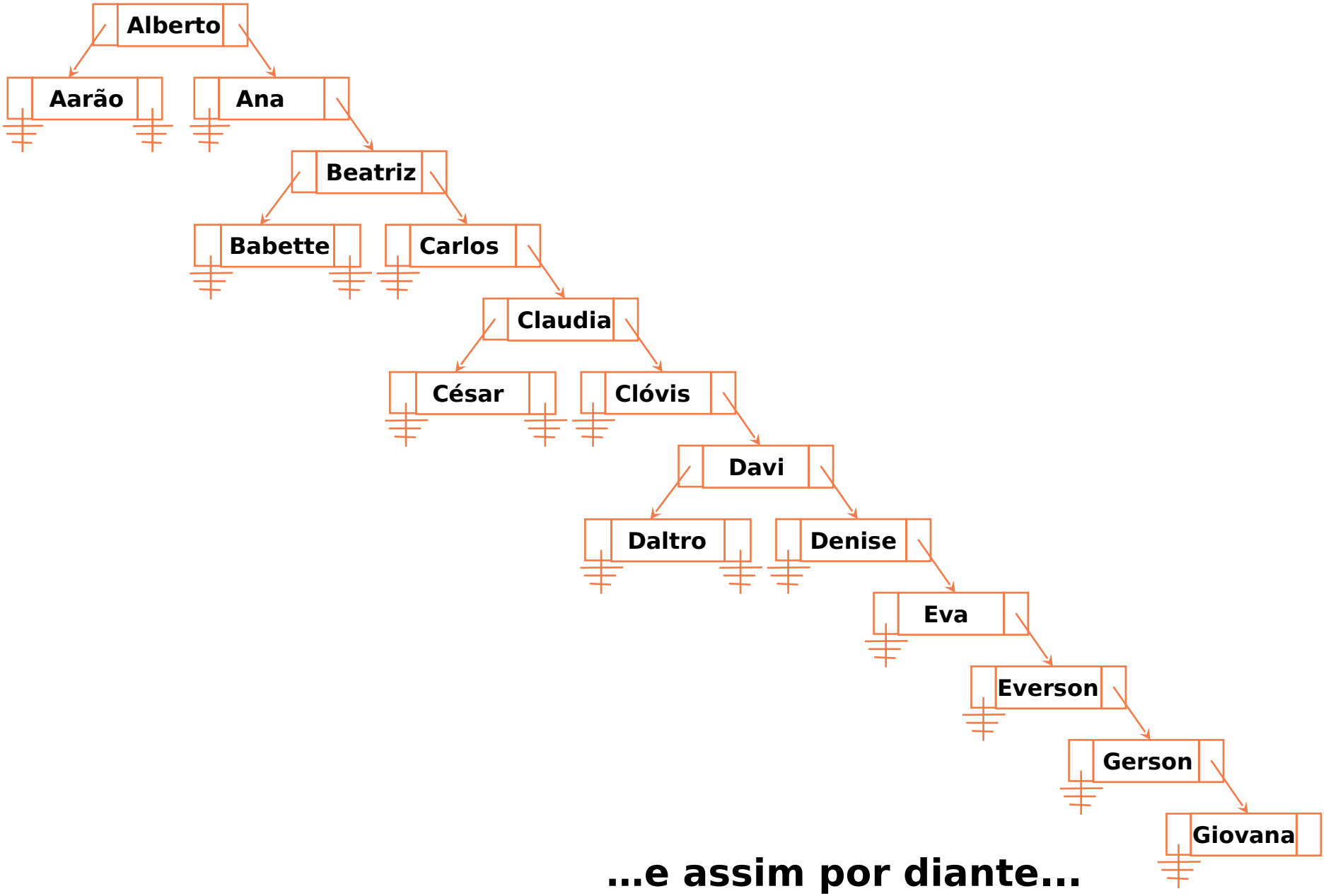
- Deterioração:
  - quando inserimos utilizando a inserção simples, dependendo da distribuição de dados, pode haver deterioração;
  - árvores deterioradas perdem a característica de eficiência de busca.

Problemas com árvores binárias de busca



Exemplo: inserção de uma série de nomes parcialmente ordenados em uma árvore binária de busca

- Alberto
- Aarão
- Ana
- Antonio
- Beatriz
- Babette
- Carlos
- Claudia
- Clóvis
- Cesar
- Davi
- Denise
- Daltro
- Eva
- Everson
- Gerson
- Giovana
- Josi
- Jaci
- Xisto
- Xênia
- Zenaide



...e assim por diante...

Exercícios

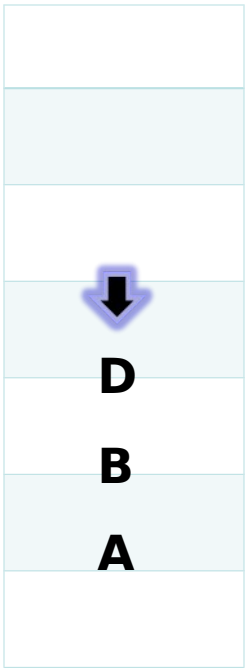
- Implemente uma classe NoBinario para representar a sua árvore
- Implemente a arvore usando Templates
- Use as melhores práticas de orientação a objetos
- Documente todas as classes, métodos e atributos.
- Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados.

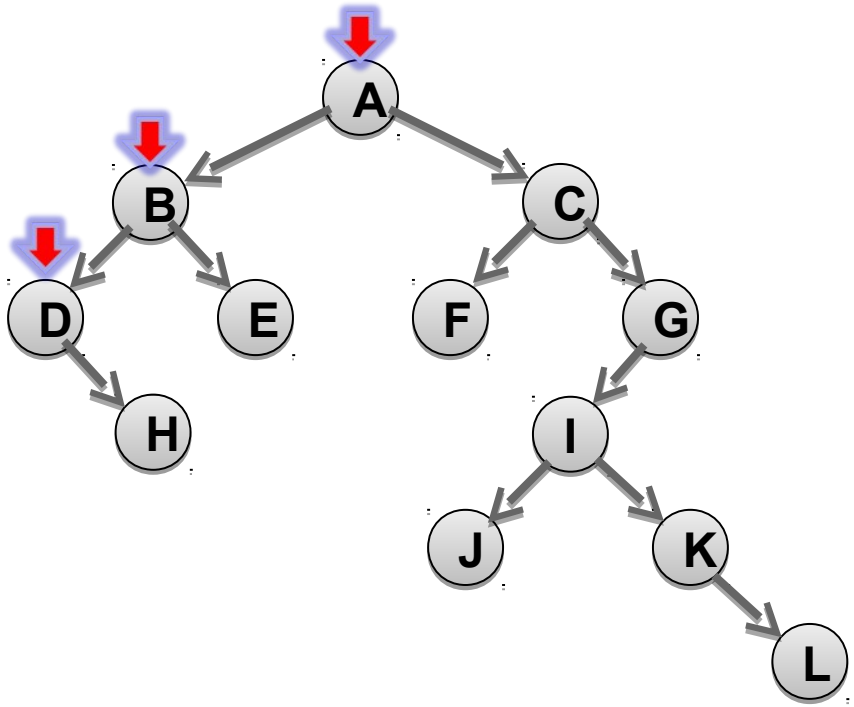
Desafio

- Implemente os percursos na versão recursiva e também iterativa:
  - **Percursos em Árvores Iterativos** são realizados através do controle do nodo que visitamos com uso de uma pilha.
  - Existem muitas maneiras de se o fazer. O algoritmo adiante é o mais simples
  - Existem outras formas mais intuitivas onde se marca se um nodo já foi visitado ou se inclue na pilha a informação se o percurso foi para a esquerda ou direita.

Percurso Emordem Iterativo

1.Crie uma pilha e monte a árvore.
2.Empilhe nodo e ache o filho da esquerda até NULO

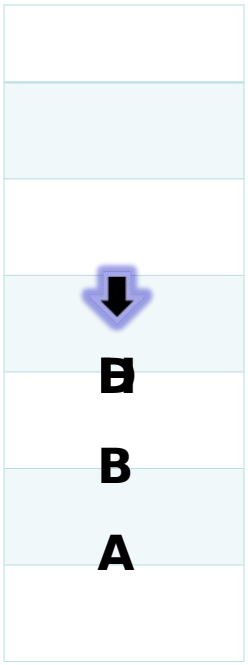


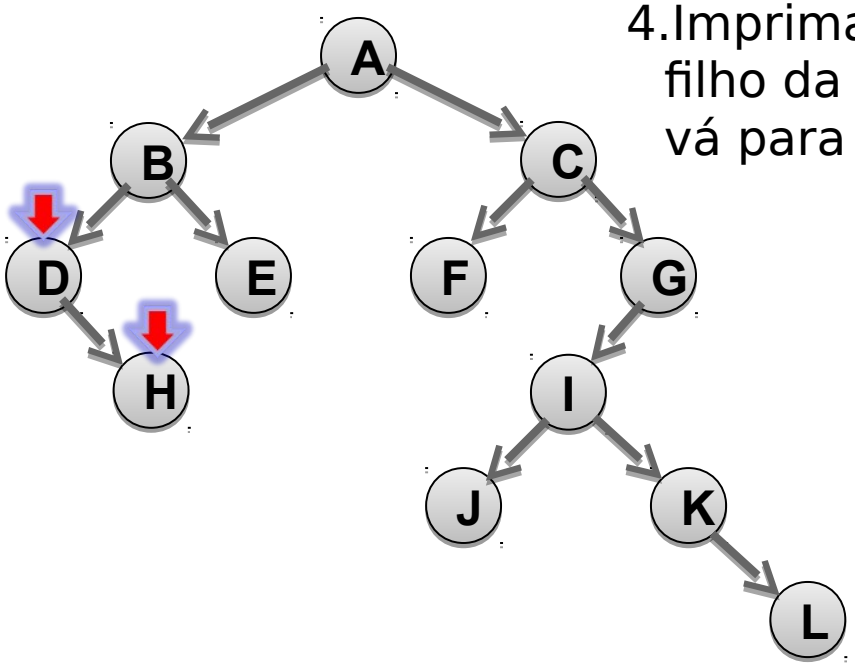


Percurso Emordem Iterativo

D

1.Crie uma pilha e monte a árvore.
2.Empilhe nodo e ache o filho da esquerda até NULO
3.Se o nodo for NULO, desempilhe.
4.Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2

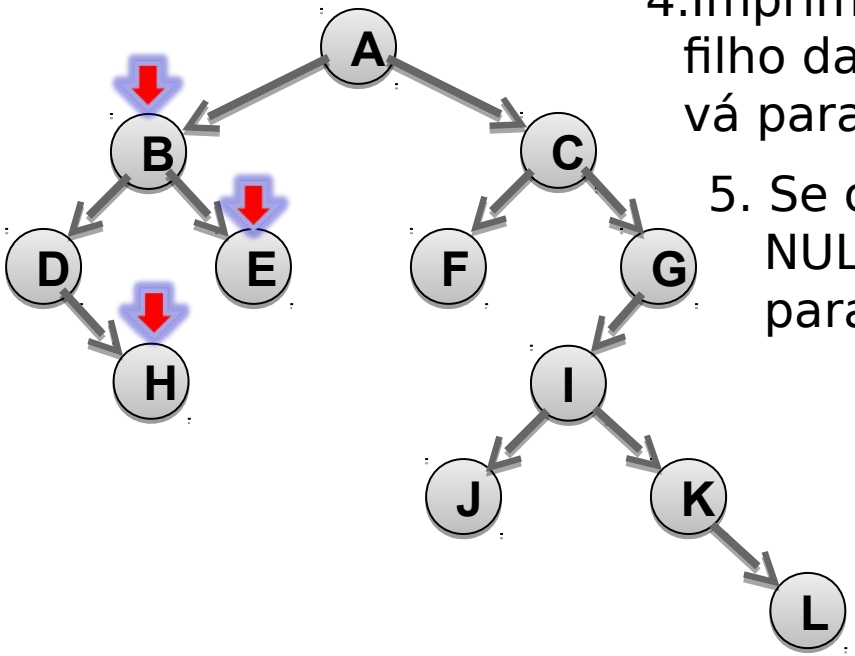


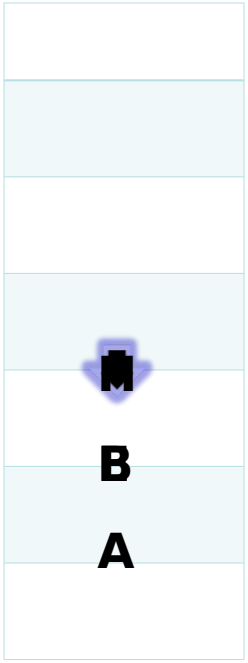


Percurso Emordem Iterativo

D H B

1. Crie uma pilha e monte a árvore.
2. Empilhe nodo e ache o filho da esquerda até NULO
3. Se o nodo for NULO, desempilhe.
4. Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4

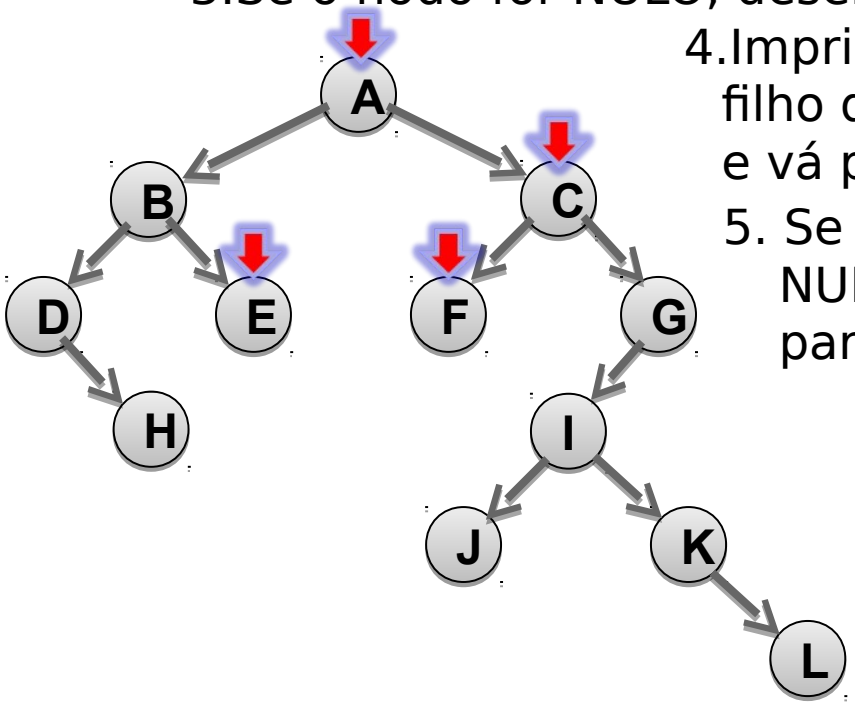


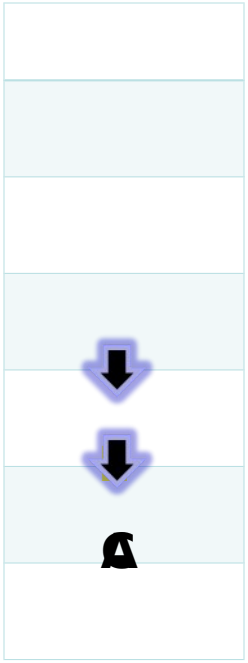


Percurso Emordem Iterativo

D H B E A

1. Crie uma pilha e monte a árvore.
2. Empilhe nodo e ache o filho da esquerda até NULO
3. Se o nodo for NULO, desempilhe.
4. Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4

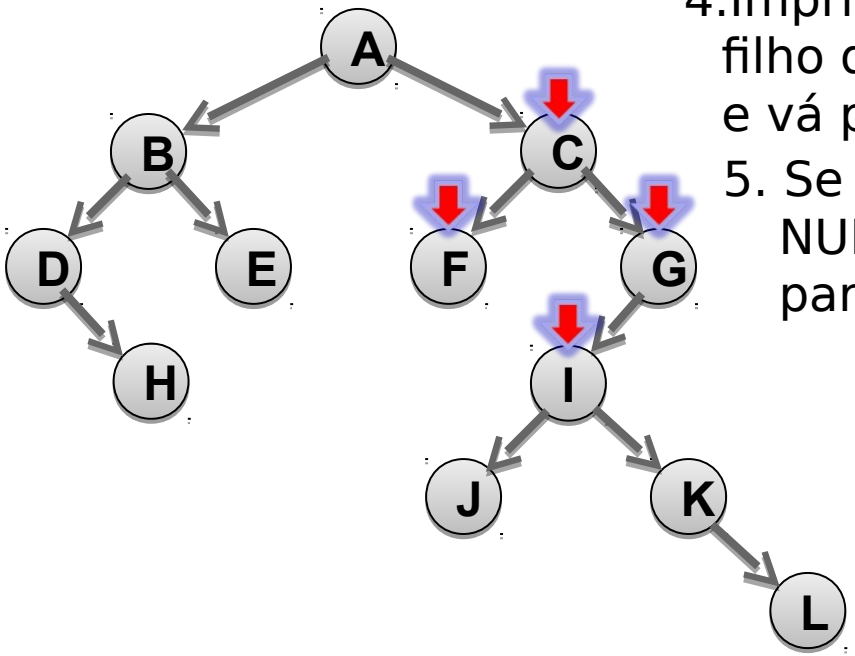


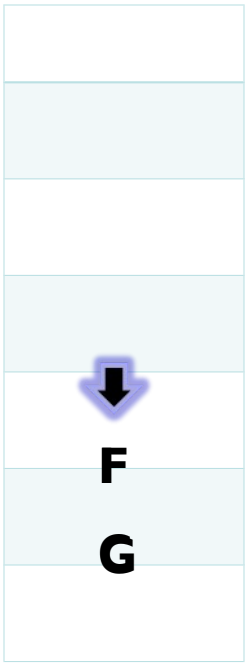


Percurso Emordem Iterativo

D H B E A F C

1. Crie uma pilha e monte a árvore.
2. Empilhe nodo e ache o filho da esquerda até NULO
3. Se o nodo for NULO, desempilhe.
4. Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4

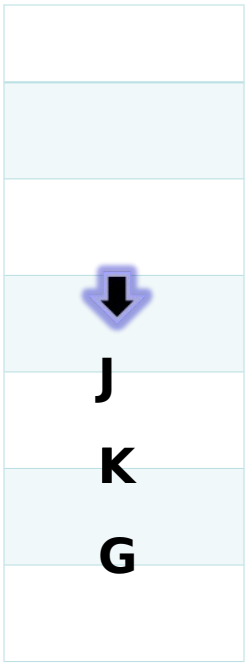


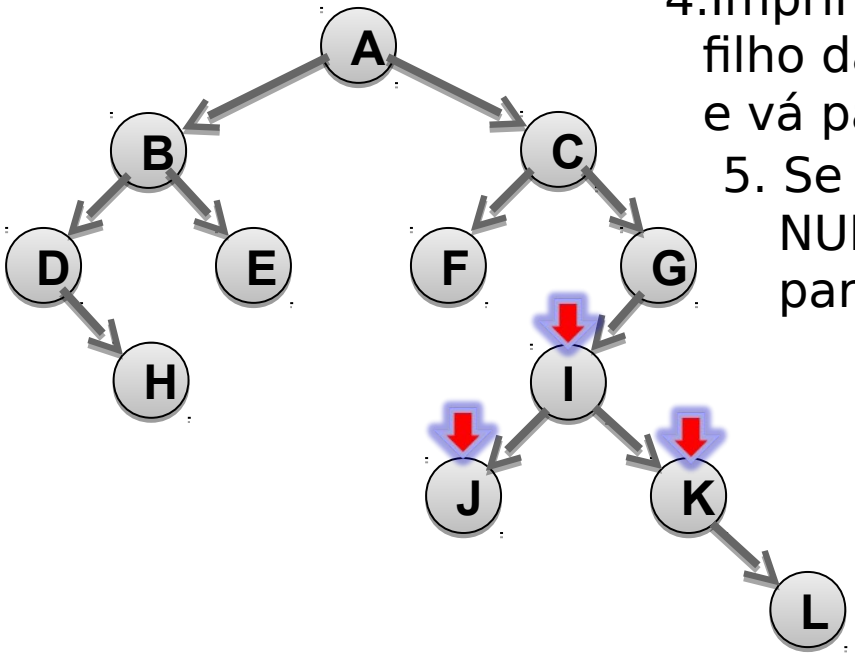


Percurso Emordem Iterativo

D H B E A F C J I

1.Crie uma pilha e monte a árvore.
2.Empilhe nodo e ache o filho da esquerda até NULO
3.Se o nodo for NULO, desempilhe.
4.Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4

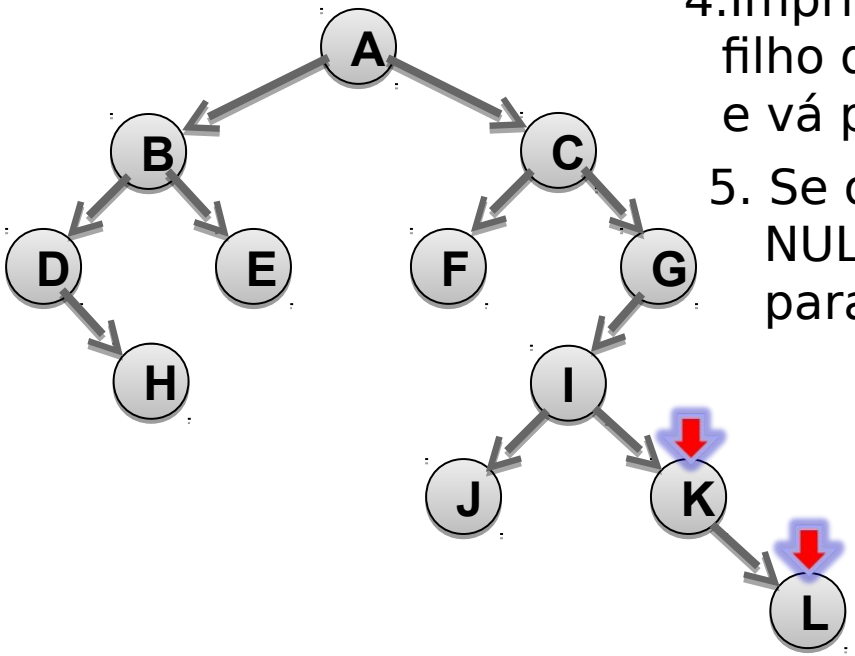




Percurso Emordem Iterativo

D H B E A F C J I K L

1. Crie uma pilha e monte a árvore.
2. Empilhe nodo e ache o filho da esquerda até NULO
3. Se o nodo for NULO, desempilhe.
4. Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4



Percurso Emordem Iterativo

D H B E A F C J I K L G

1. Crie uma pilha e monte a árvore.
2. Empilhe nodo e ache o filho da esquerda até NULO
3. Se o nodo for NULO, desempilhe.
4. Imprima o nodo e ache filho da direita (empilhe) e vá para passo 2
5. Se o filho da direita for NULO, desempilhe e vá para o passo 4
6. Se o filho da direita for NULO e a pilha estiver vazia, pare.