Estruturas de Dados

A. G. Silva, A. von Wangenheim, J. E. Martina

Revisado em 26 de setembro de 2018

Introdução

Árvores

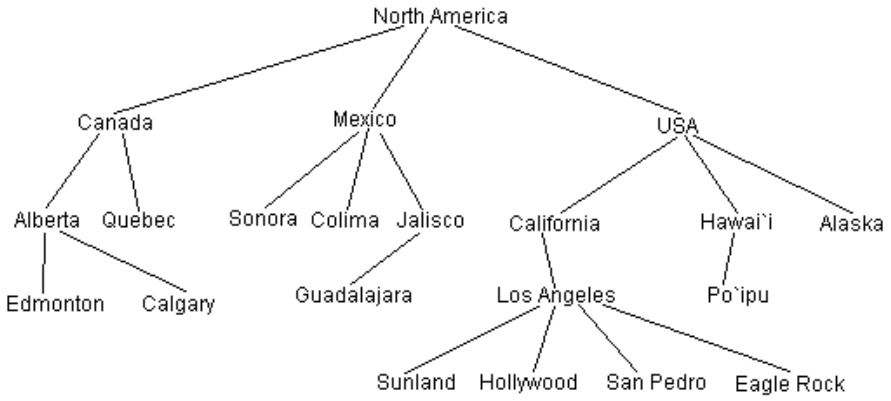
Estruturas de dados que se caracterizam por uma organização hierárquica de seus elementos. Essa organização permite a definição de algoritmos relativamente simples, recursivos e mais eficientes.

Introdução

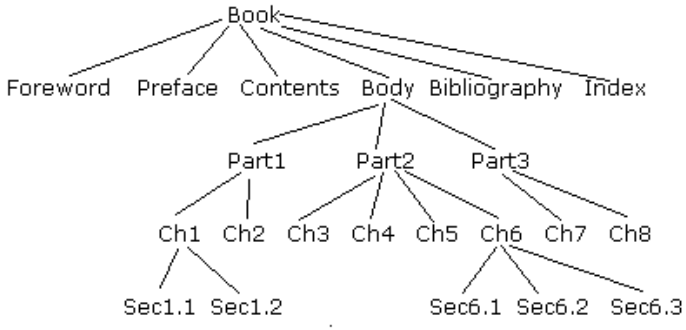
- No cotidiano, diversas informações são organizadas de forma hierárquica;
- Como exemplo, podem ser citados:
  - O organograma de uma empresa;
  - A divisão de um livro em capítulos, seções, tópicos;
  - A árvore genealógica de uma pessoa.

Introdução

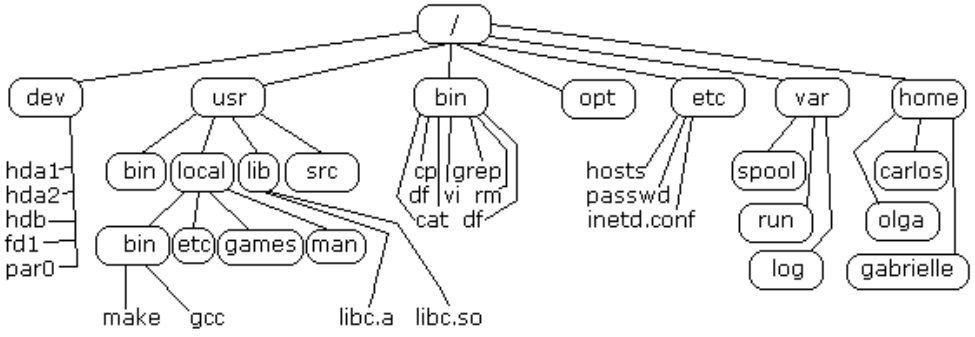
Exemplo 1



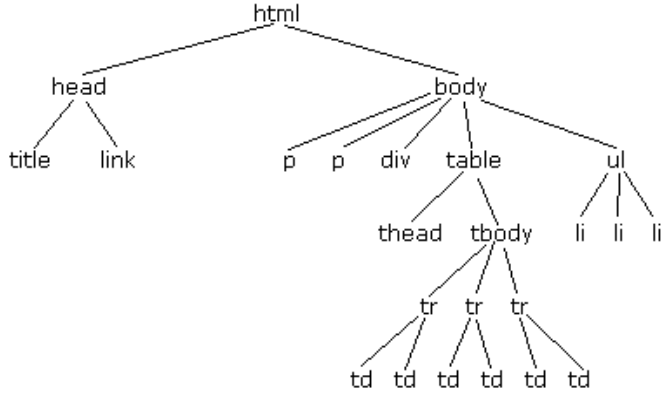
Exemplo 2



Exemplo 3



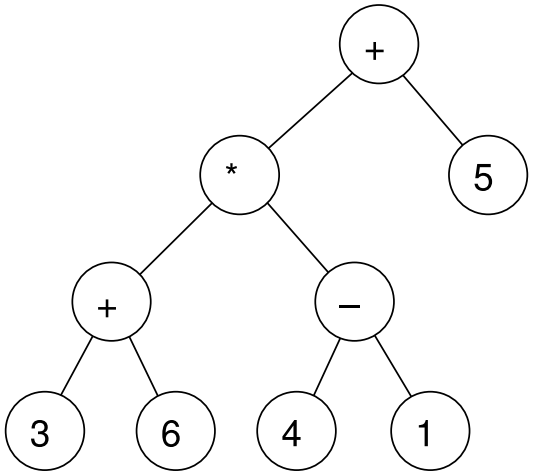
Exemplo 4



Fonte: http://cs.lmu.edu/~ray/notes/orientedtrees/

Introdução

- Árvores binárias representando expressões aritméticas
  - nodos folhas representam operandos
  - nodos internos representam operadores
  - exemplo: (3+6)*(4-1)+5

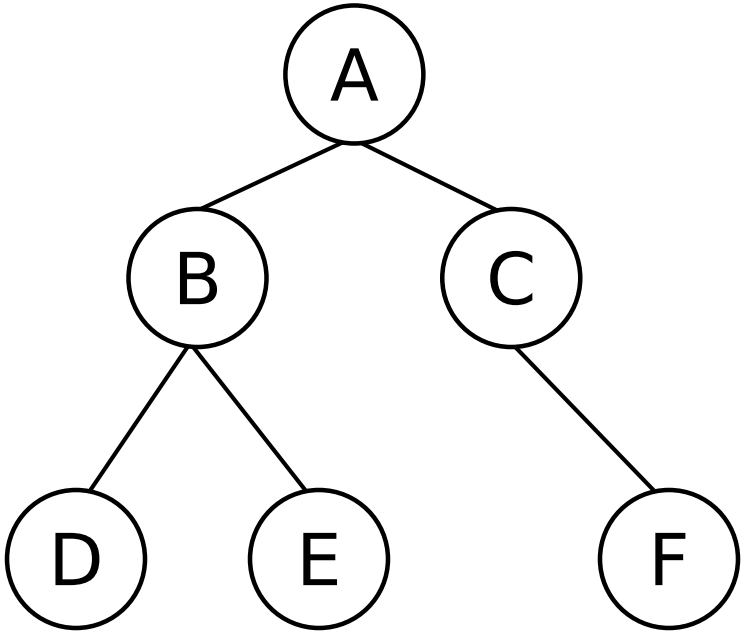


Introdução

- De um modo mais formal, pode-se 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 $n \geq 0$ conjuntos disjuntos $c_1, c_2, ..., c_n$, sendo que cada um desses conjuntos também é uma árvore (denominada subárvore).

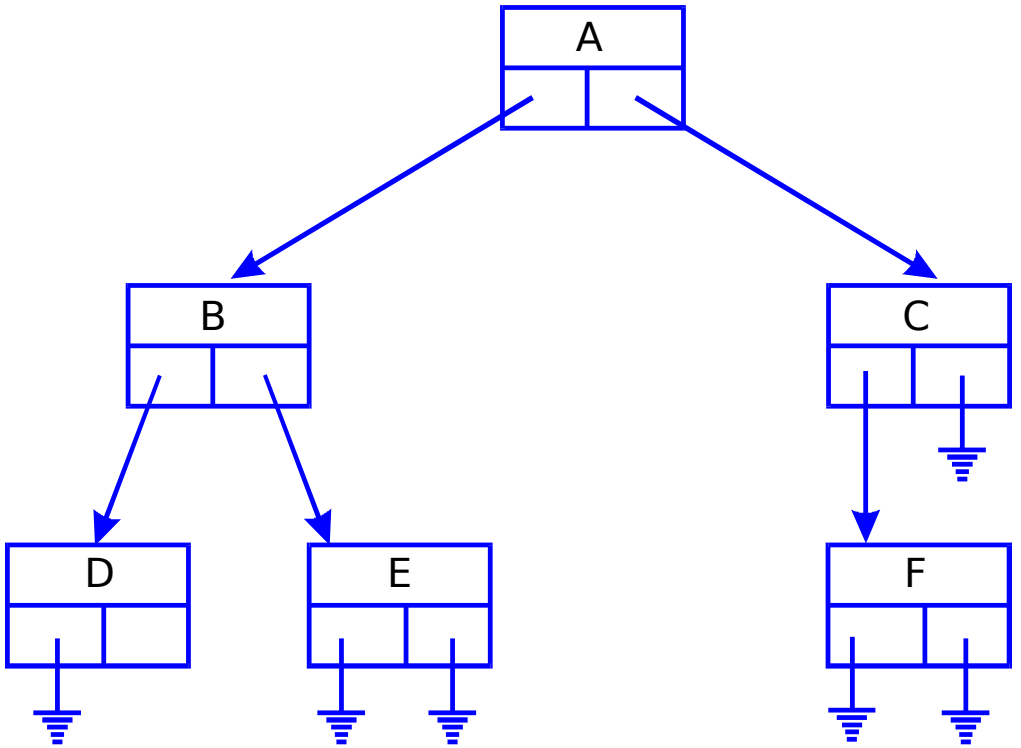
Representações

- Representação hierárquica



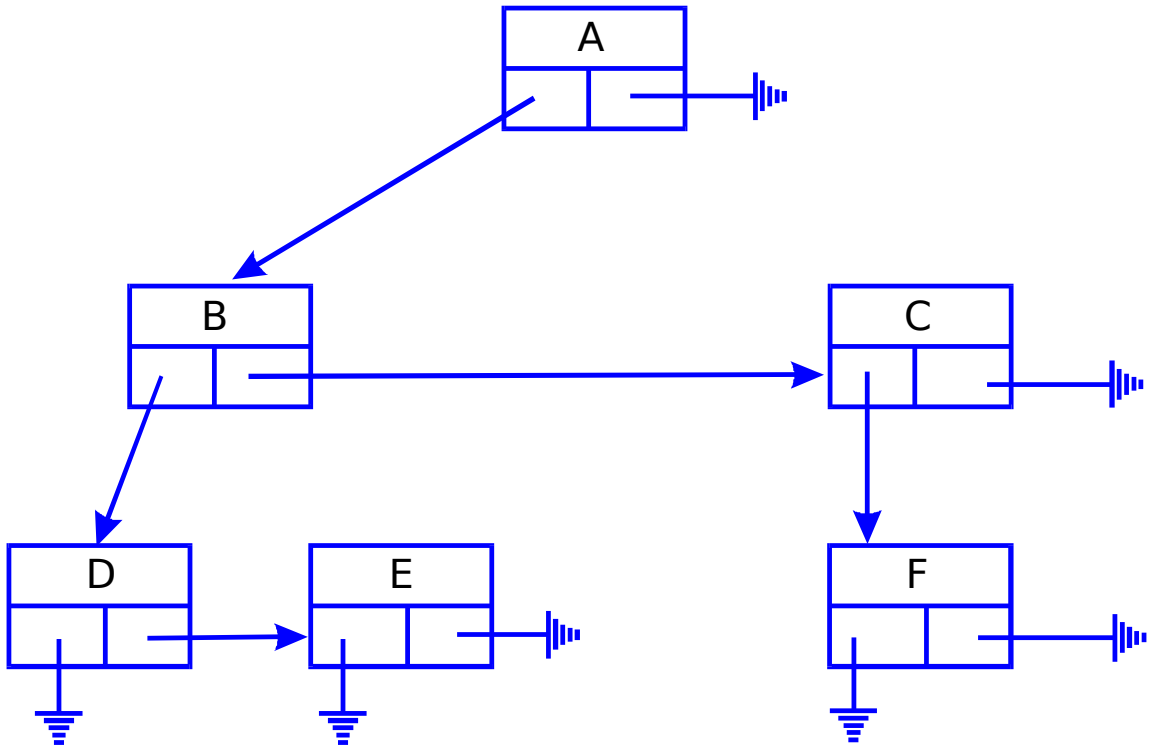
Representações

- Representação hierárquica apenas para **árvores binárias**
  - nodo com ponteiro para seu **filho esquerdo** e para seu **filho direito**



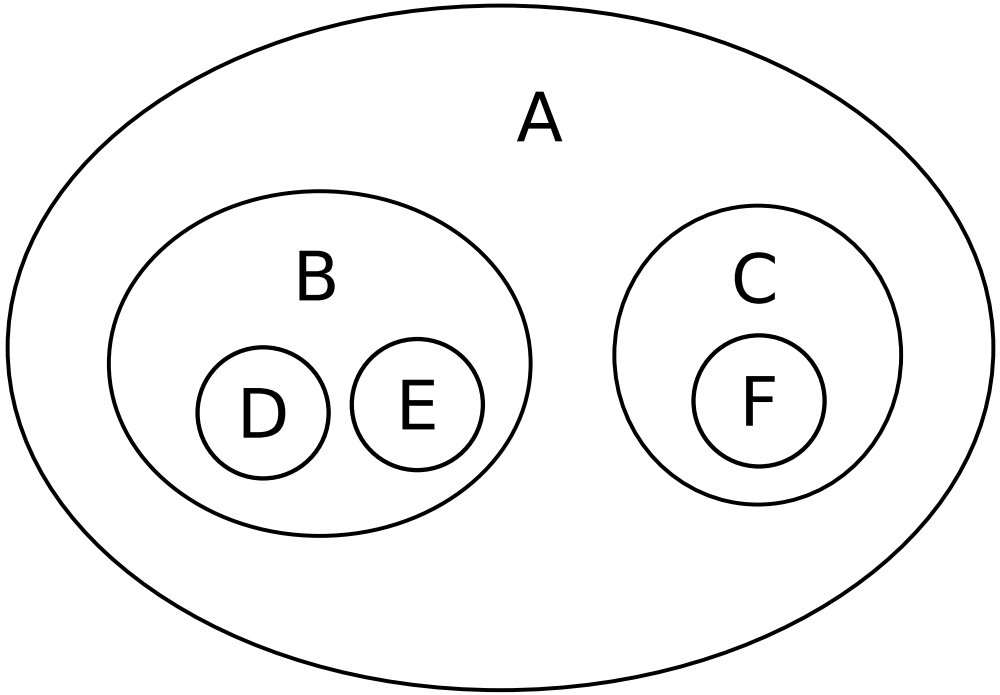
Representações

- Representação hierárquica **genérica** – “lista de filhos”
  - nodo com ponteiro para seu **primeiro filho** e para o **próximo irmão**



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

- Representação por tabela
  - Cada nodo é seguido por um número que representa o índice de seu pai.

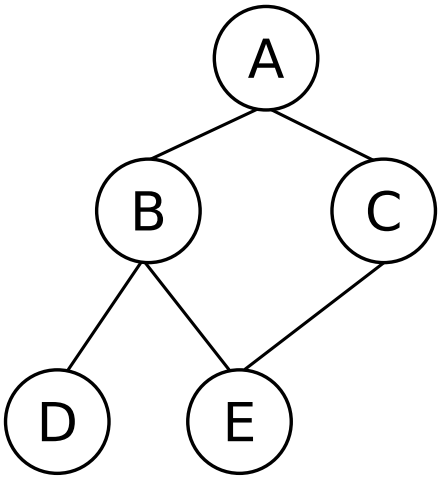
1	–	
2	C	4
3	F	2
4	A	1
5	E	6
6	B	4
7	D	6

Representações

- As três primeiras representações permitem uma melhor visualização das árvores ou construção dessas estruturas em memória principal;
- As três últimas, por sua vez, facilitam a persistência dos nodos das árvores (em arquivos, por exemplo), possibilitando assim a sua reconstrução.

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 arestas orientadas no sentido “pai-*para*-filho” intermediárias entre A e B;
- Existe sempre um caminho entre a raiz e qualquer nodo da árvore;

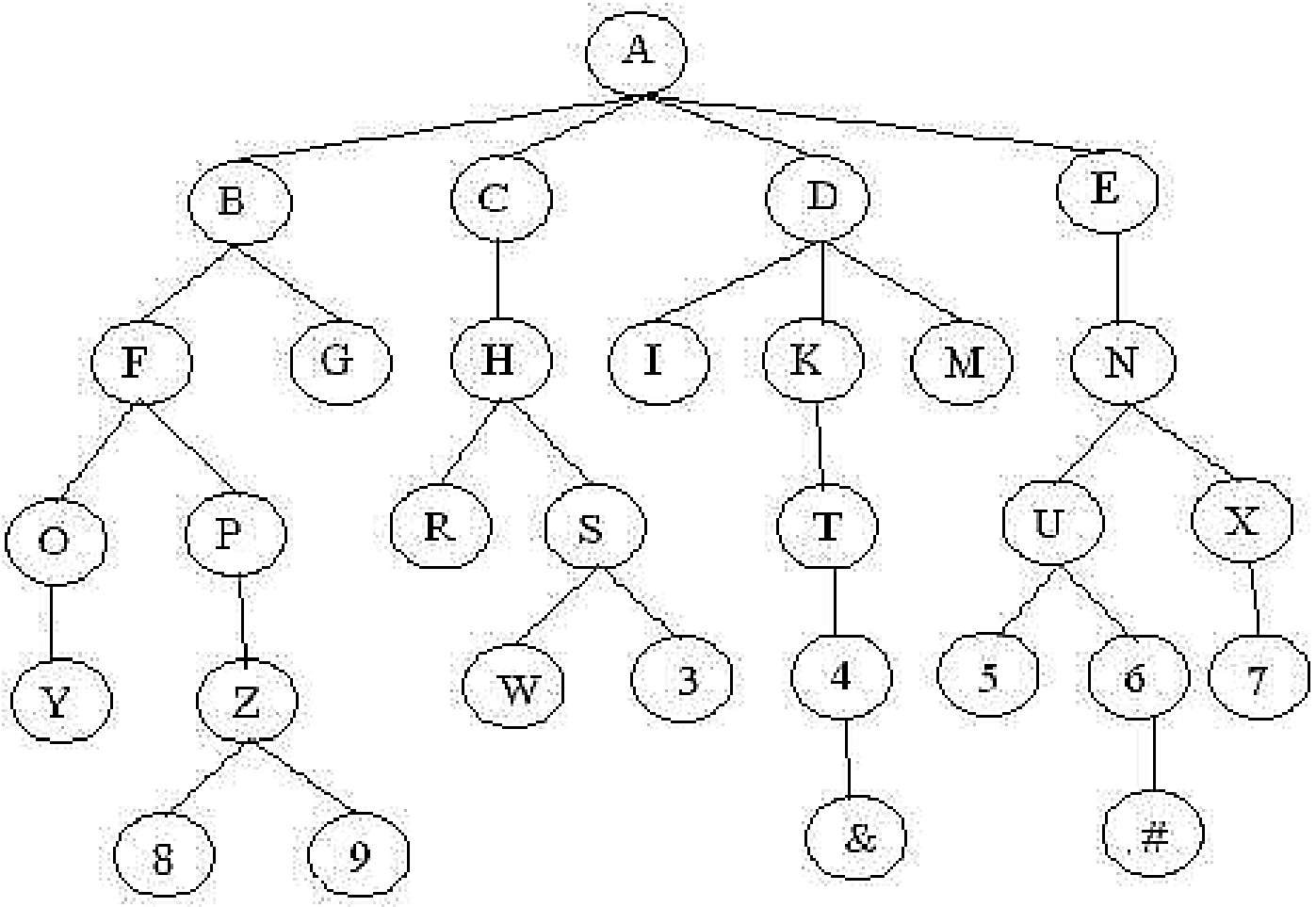
Definições

- 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 está no nível 0; seus descendentes diretos estão no nível 1, e assim por diante;
- **Grau da árvore**: é igual ao grau do nodo de maior grau da árvore;
- **Comprimento do caminho**: quantidade de arestas em um dado caminho;
- **Altura da árvore**: comprimento do caminho mais longo da raiz até uma das folhas; se árvore tiver um único nodo: altura é zero; se for vazia: altura é -1 por definição.

Exercício



Exercício

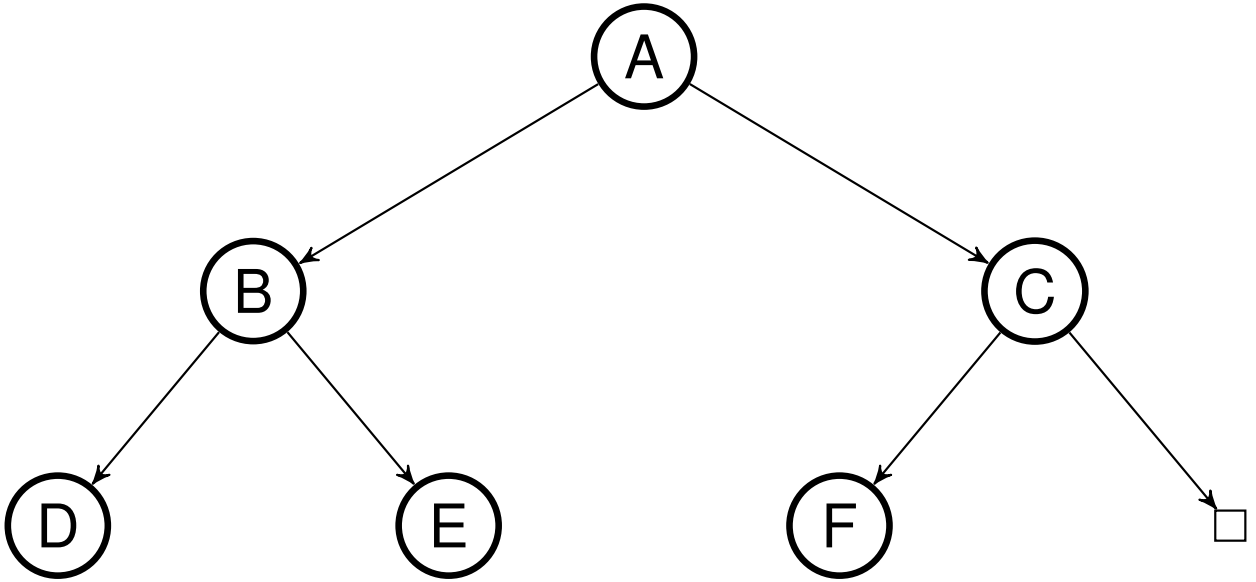
- Qual é a raiz da árvore?
- Quais são os nodos terminais?
- Qual o grau da árvore?
- Qual a altura da árvore?
- Quais são os nodos descendentes do nodo D?
- Quais são os nodos ancestrais do nodo #?
- Os nodos 4 e 5 são nodos irmãos?
- Há caminho entre os nodos C e S?
- Qual o nível do nodo 5?
- Qual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

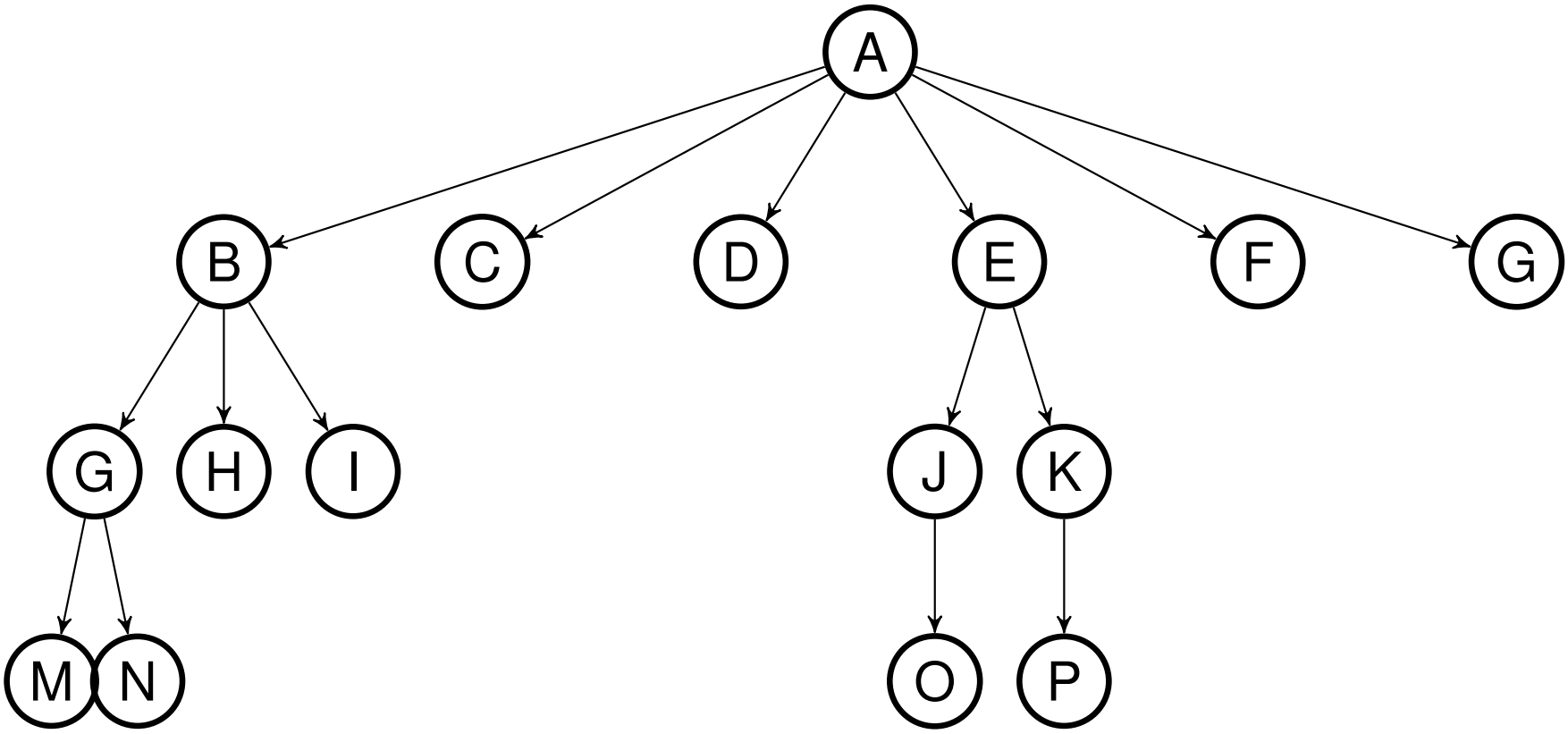
- Exemplo de árvore binária;



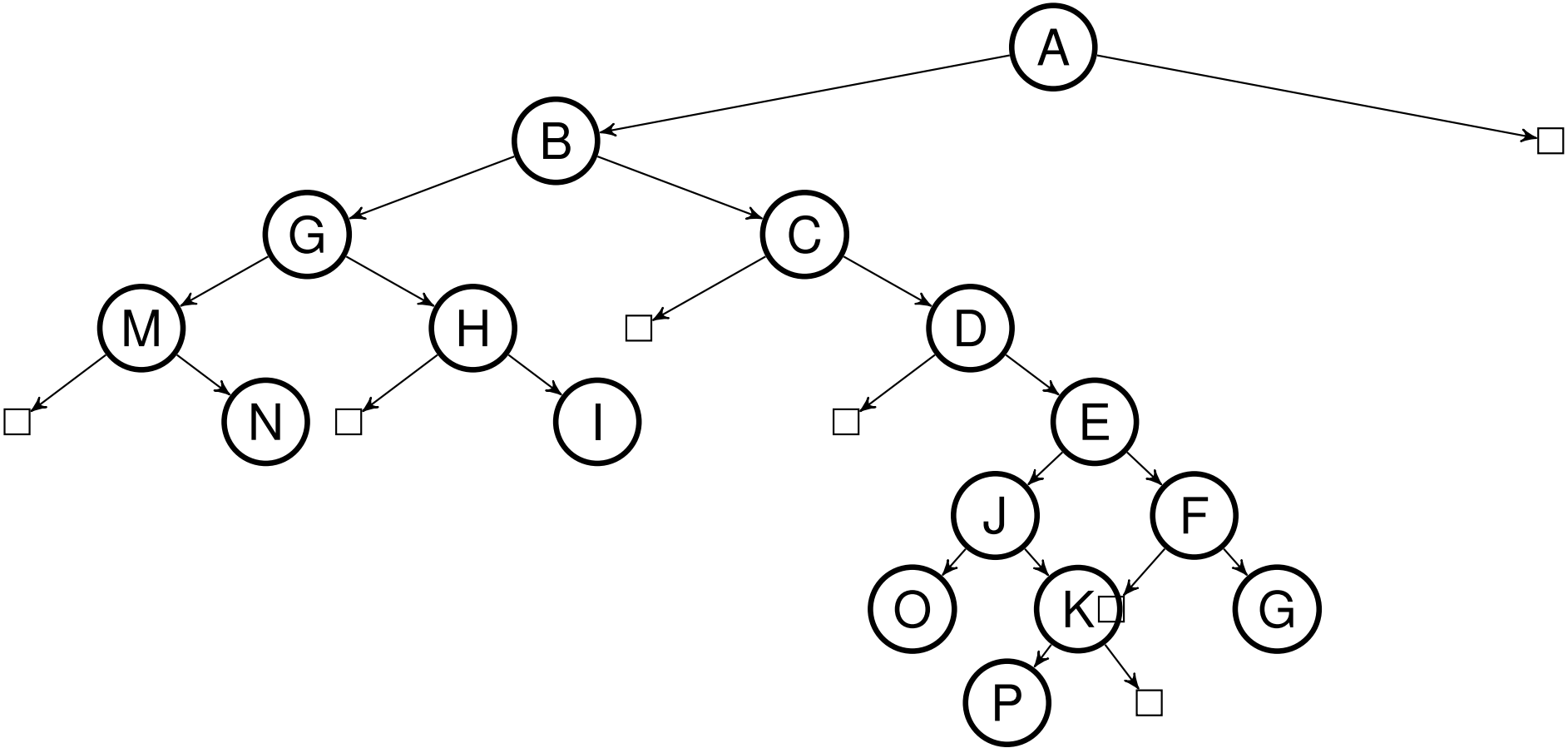
Transformações

- 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 B, 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.

Árvores N-ária



Árvores Binária para representar uma N-ária



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 *filhoAEsquerda;
 tNodo *filhoADireita;
 tInfo 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 $nodosAEsquerda = numeroDeNodos/2$ nodos, usando este mesmo procedimento;
  - Gere a subárvore direita com $nodosADireita = numeroDeNodos – nodosAEsquerda - 1$ nodos, usando este mesmo procedimento.

Árvore Binária Balanceada

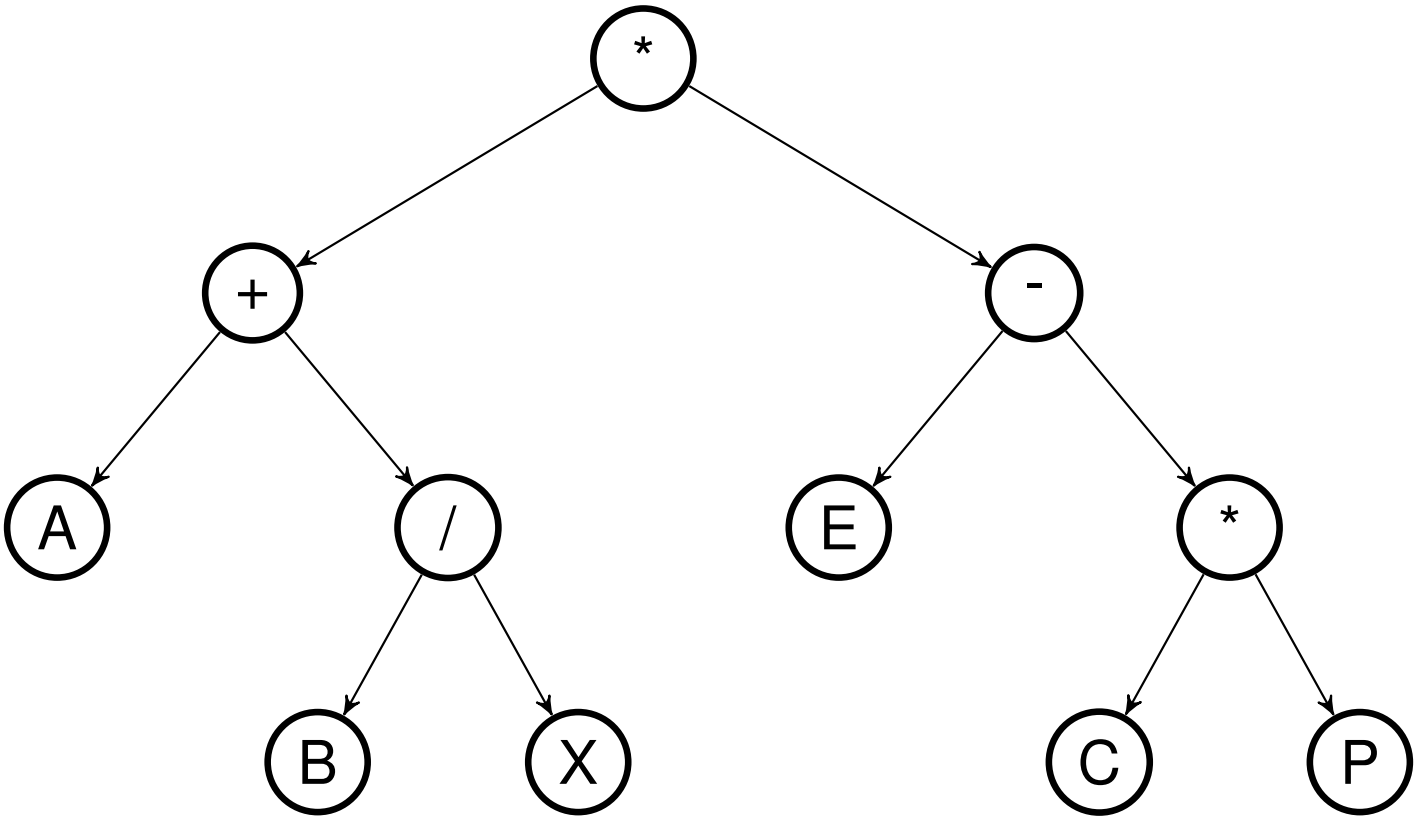
```
tNodo* constroiArvore(inteiro numeroDeNodos)
 inteiro nodosAEsquerda, nodosADireita;
 tInfo info;
 tNodo *novoNodo;
 inicio
  se numeroDeNodos = 0 entao
   retorna NULO;
  nodosAEsquerda ← numeroDeNodos / 2;
  nodosADireita ← numeroDeNodos - nodosAEsquerda - 1;
  aloque(info);
  ler(info);
  aloque(novoNodo);
  novoNodo->info ← info;
  novoNodo->filhoAEsquerda ← constroiArvore(nodosAEsquerda);
  novoNodo->filhoADireita ← constroiArvore(nodosADireita);
  retorna novoNodo;
 fim
```

Percursos em Árvores Binárias

- O percurso em árvores binárias corresponde à visitação (processamento representado, por exemplo, por um inserção em uma lista) de cada um dos nodos da estrutura:
  - Partimos de um nodo inicial (raiz) e visitamos todos os demais nodos recursivamente em uma ordem previamente especificada;
- Como exemplo, considere uma árvore binária utilizada para representar uma expressão (com as seguintes restrições):
  - Cada operador binário representa uma bifurcação;
  - Seus dois operandos correspondentes são representados por suas subárvores.

Percursos em Árvores Binárias

Expressão: (A + (B / X)) * (E - (C * P))



Percursos 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;
  - PosOrdem(e,d,r) – Postorder.

Percursos 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;
  - PosOrdem(e,d,r): visite a raiz DEPOIS das subárvores;
- As subárvores são SEMPRE visitadas da esquerda para a direita.

Percursos em Árvores Binárias

- Se percorrermos a árvore anterior usando as ordens definidas, teremos as seguintes seqüências:
  - Preordem (notação prefixada) : * + A / B X - E * C P
  - Emordem (notação infixada) : A + B / X * E - C * P
  - Pósordem (notação posfixada) : A B X / + * E C P - *

Percurso em Pré-Ordem

```
PreOrdem(tNodo *raiz, tInfo V[])
 inicio
  se raiz ≠ NULO entao
   V.add(raiz->info);
   PreOrdem(raiz->filhoAEsquerda);
   PreOrdem(raiz->filhoADireita);
  fim se
 fim
```

Percurso em Em-Ordem

```
EmOrdem(tNodo *raiz, tInfo V[])
 inicio
  se raiz ≠ NULO entao
   EmOrdem(raiz->filhoAEsquerda);
   V.add(raiz->info);
   EmOrdem(raiz->filhoADireita);
  fim se
 fim
```

Percurso em Pós-Ordem

```
PosOrdem(tNodo *raiz, tInfo V[])
 inicio
  se raiz ≠ NULO entao
   PosOrdem(raiz->filhoAEsquerda);
   PosOrdem(raiz->filhoADireita);
   V.add(raiz->info);
  fim se
 fim
```

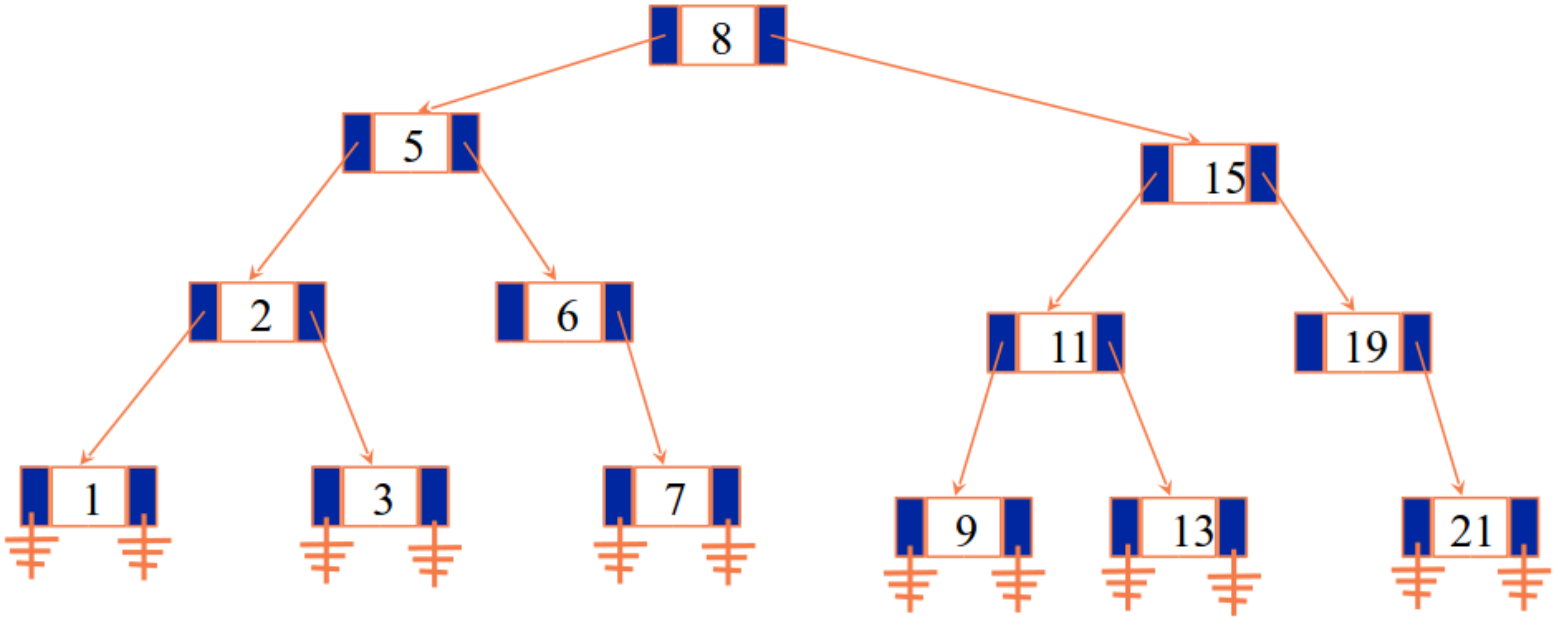
Árvores Binárias de Busca

- 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 de Árvores Binárias de Busca

- 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;
- 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 $O(\log n)$;

Exemplo de árvore binária de busca

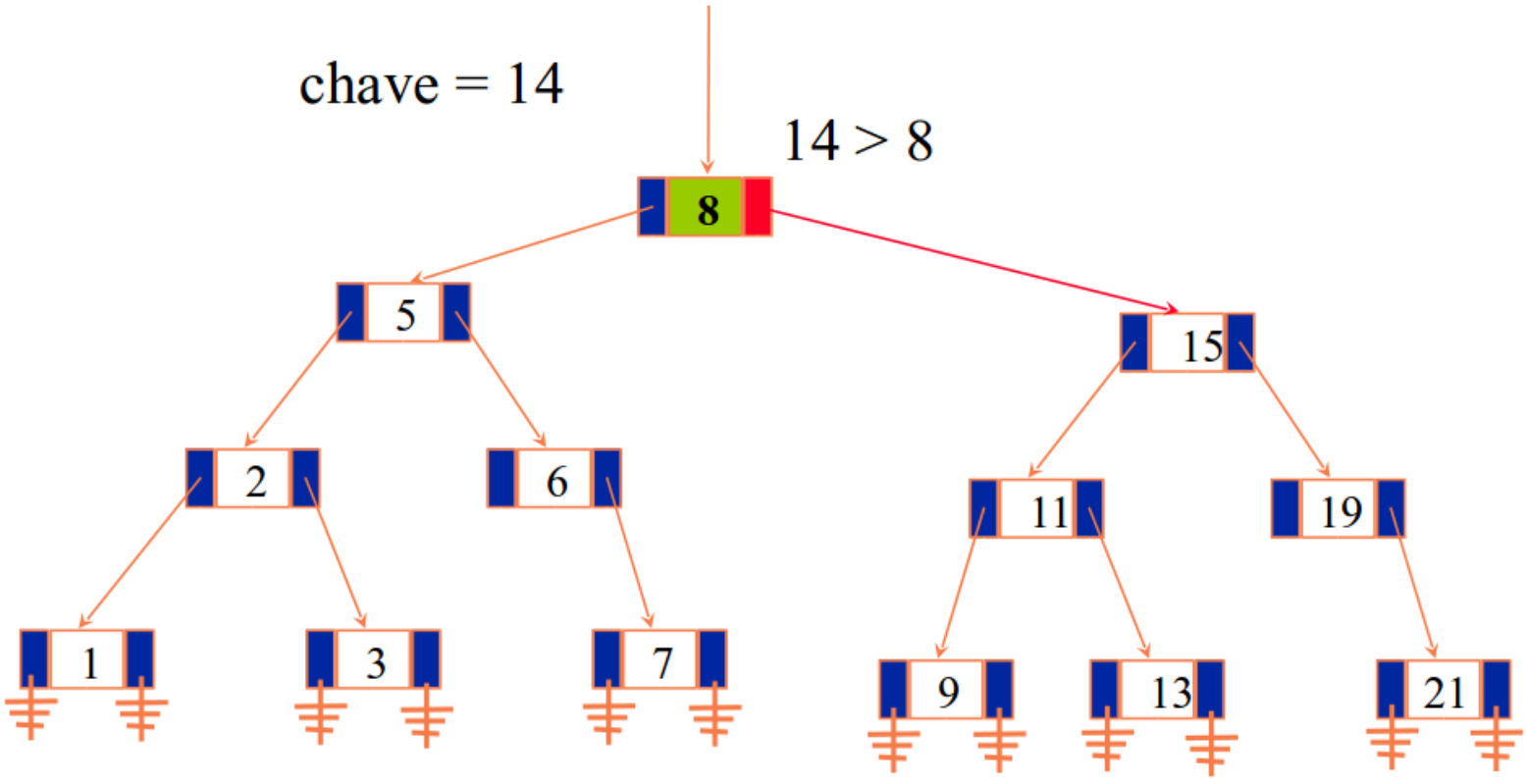


Algoritmo de Busca

```
tNodo* busca(tInfo chave, tNodo *ptr)
 inicio
  enquanto (ptr ≠ NULO
            E ptr->info ≠ chave) faca
   // esquerda ou direita.
   se (ptr->info < chave) entao
    ptr ← ptr->filhoADireita
   senao
    ptr ← ptr->filhoAEsquerda;
   fim se
  fim enquanto
  retorne ptr;
 fim
```

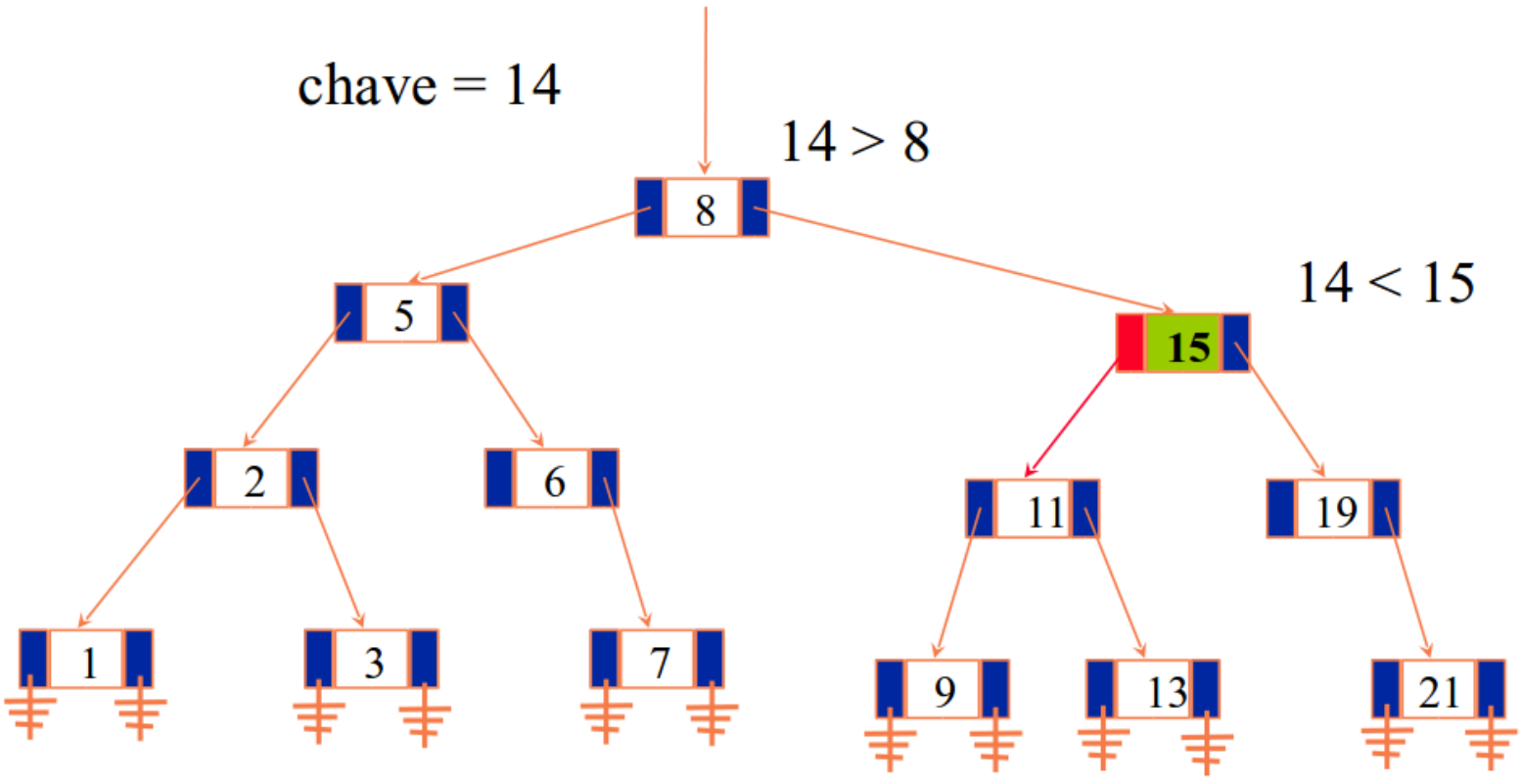
Exemplo

Inserção de um elemento com chave = 14



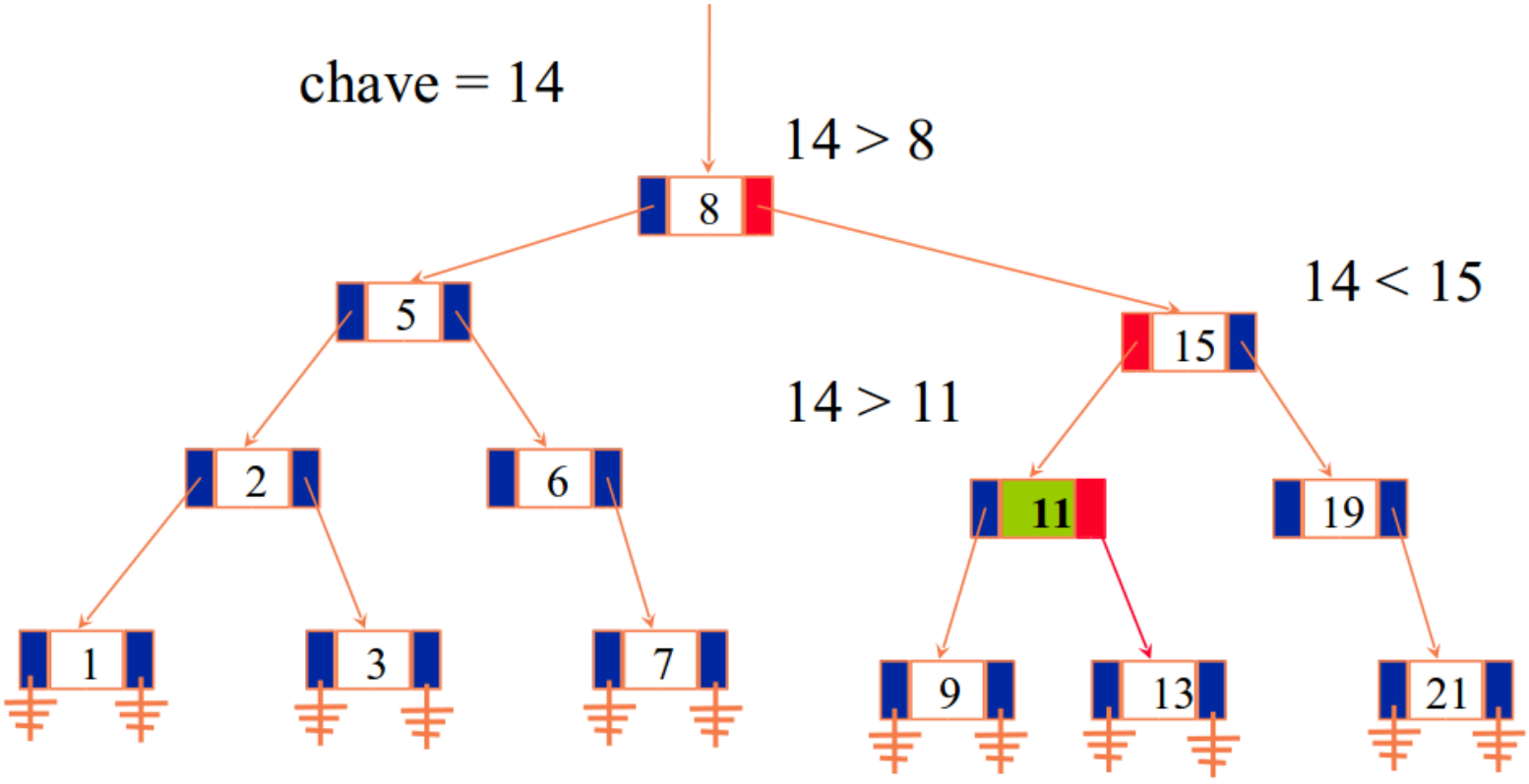
Exemplo

Inserção de um elemento com chave = 14



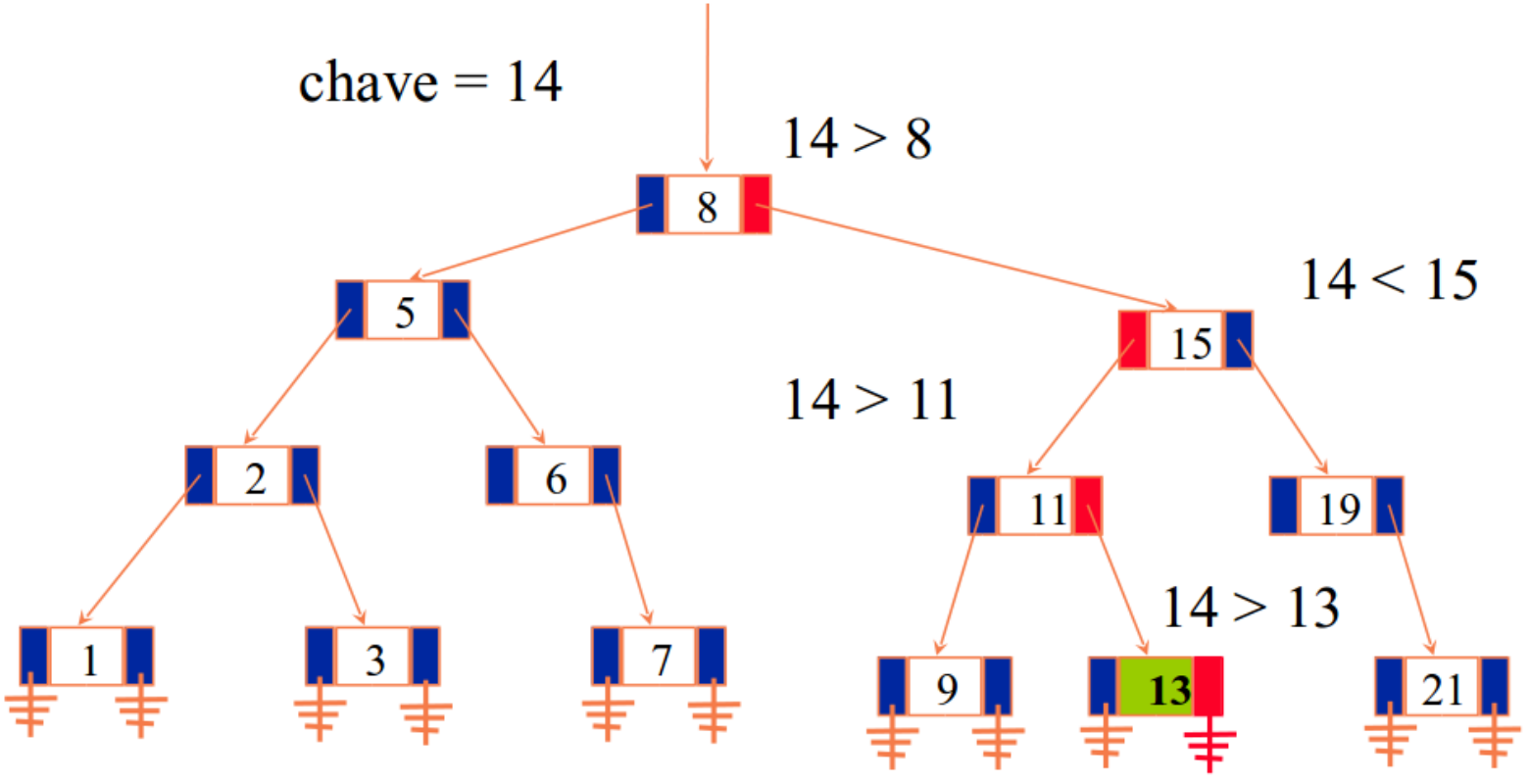
Exemplo

Inserção de um elemento com chave = 14



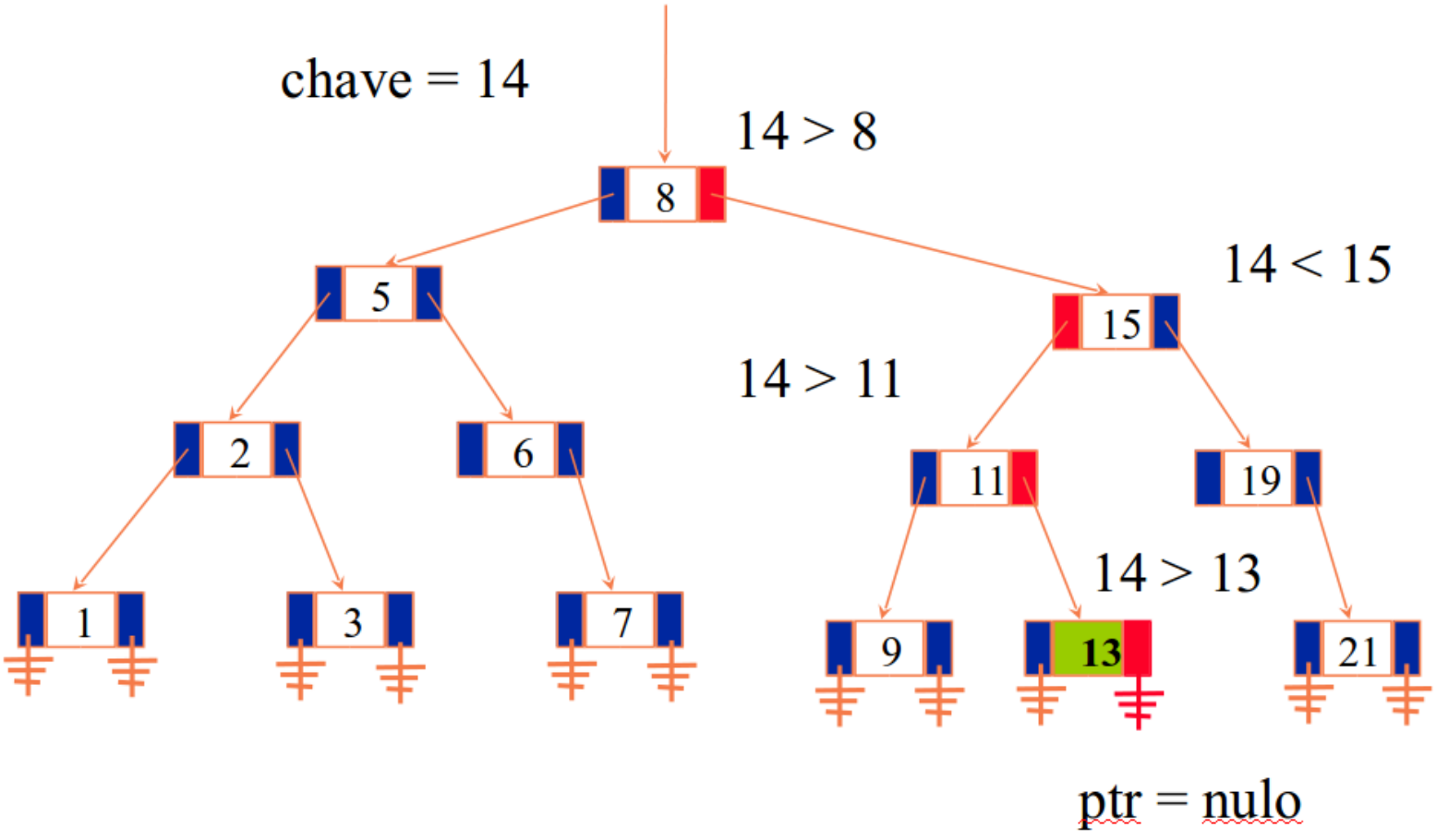
Exemplo

Inserção de um elemento com chave = 14



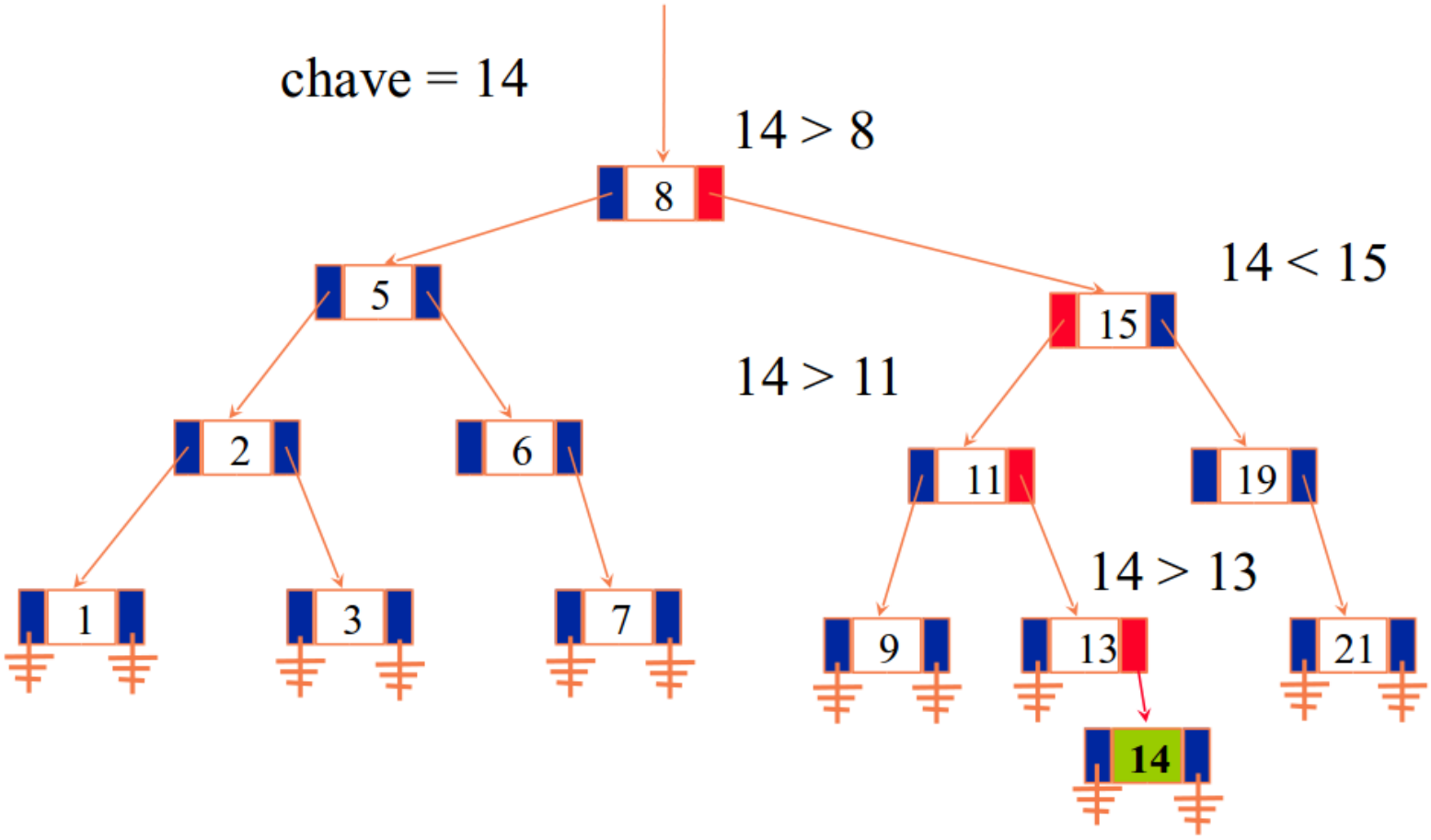
Exemplo

Inserção de um elemento com chave = 14



Exemplo

Inserção de um elemento com chave = 14



Algoritmo de Inserção

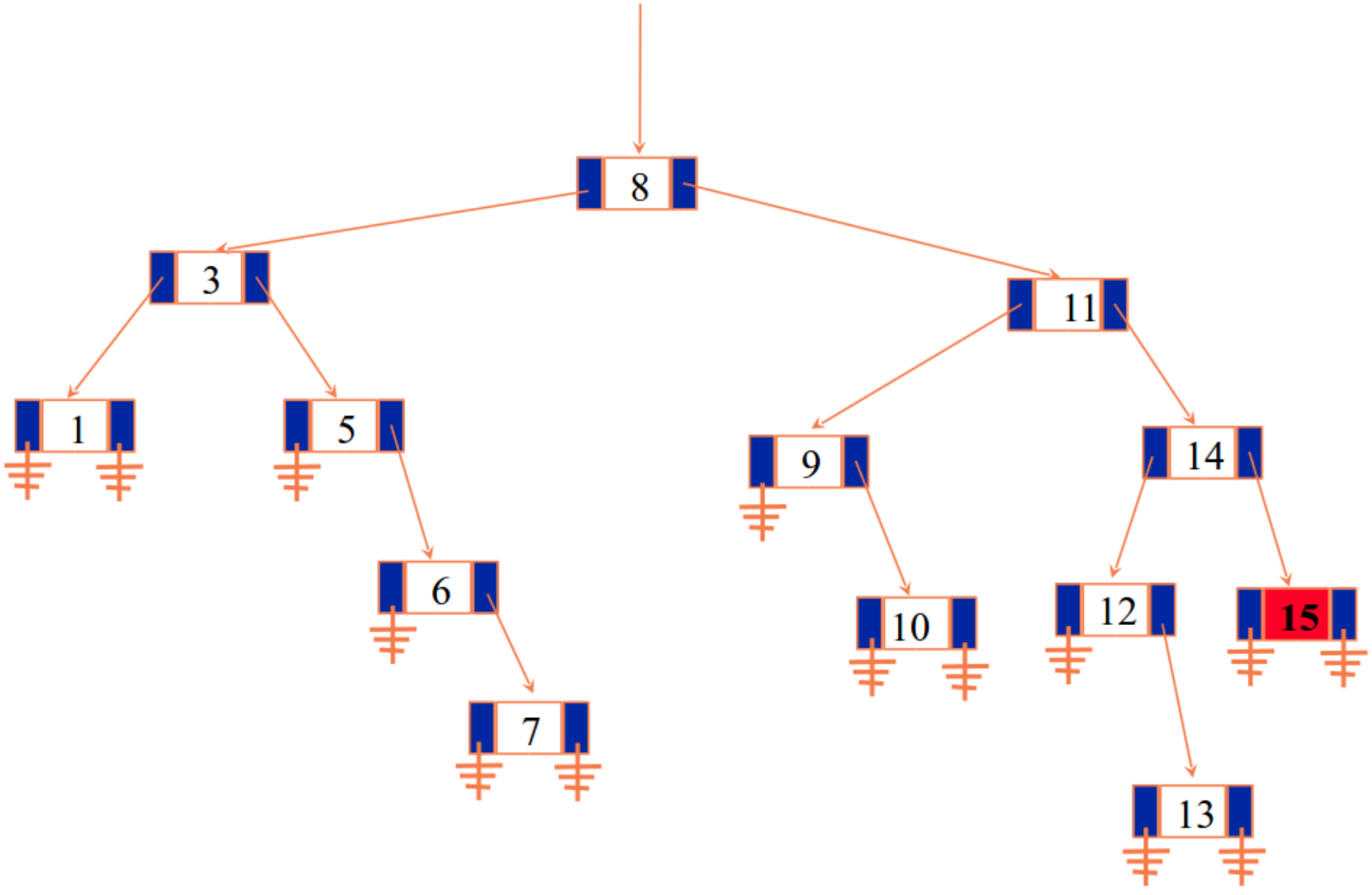
```
tNodo* insercao(tNodo *self, tInfo info)
 tNodo *oNovo;
 inicio
  se (info < self->info) entao
  // insercao 'a esquerda
   se (self->filhoAEsquerda = NULO) entao
      oNovo ← aloque(tNodo); oNovo->info ← info;
      oNovo->filhoAEsquerda ← NULO;
      oNovo->filhoADireita ← NULO;
      self->filhoAEsquerda ← oNovo;
   senao
      raiz ← insercao(self->filhoAEsquerda, info);
   fim se
   senao
      // insercao 'a direita
   se (self->filhoADireita = NULO) entao
      oNovo ← aloque(tNodo); oNovo->info ← info;
      oNovo->filhoAEsquerda ← NULO;
      oNovo->filhoADireita ← NULO;
      self->filhoADireita ← oNovo;
   senao
      raiz ← insercao(self->filhoADireita, info);
   fim se
  fim se
  retorna raiz
 fim
```

Algoritmo de Deleção

- A remoção 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 remoção deve remanejar os nodos.

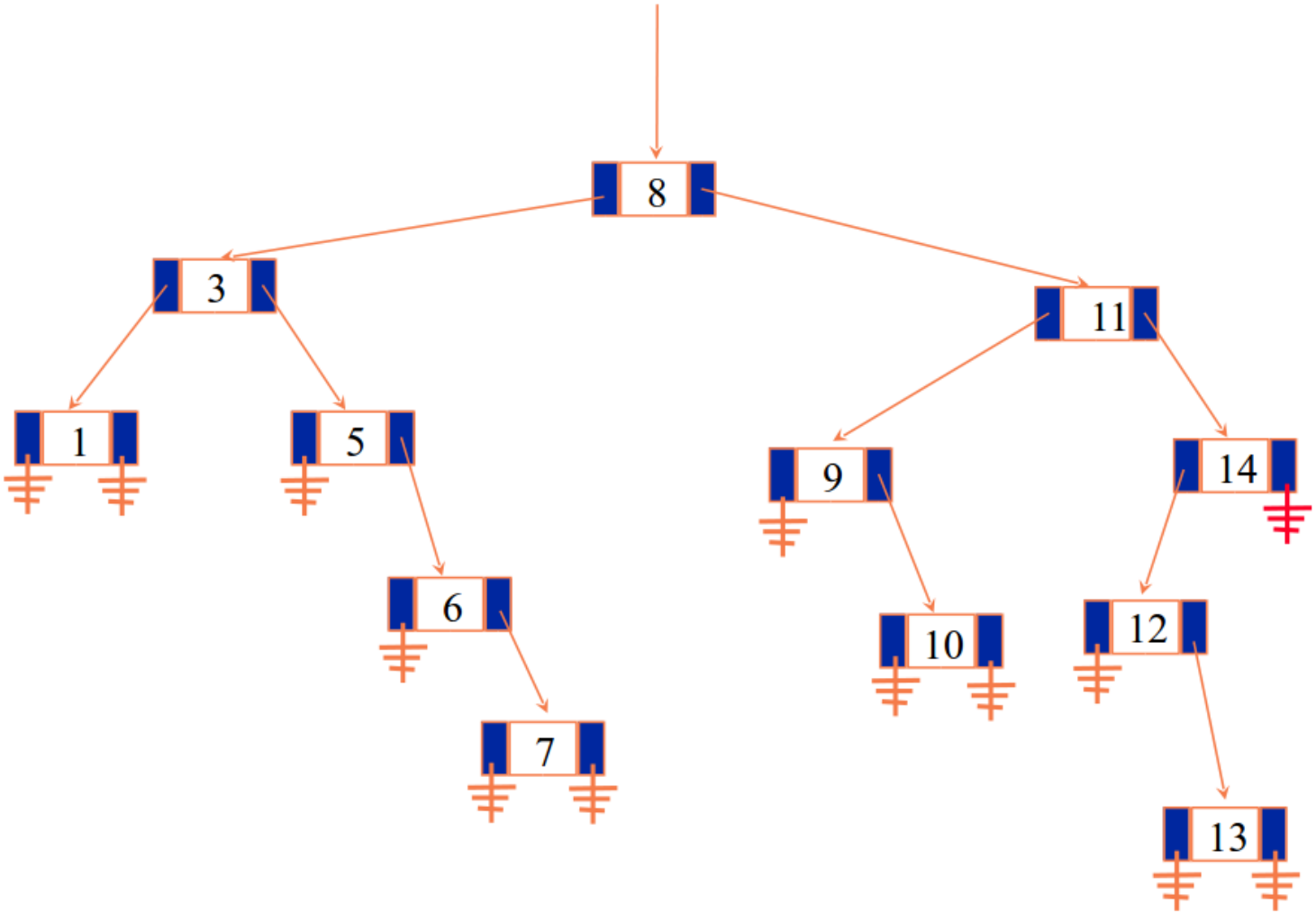
Exemplo

Remoção do nodo com chave = 15



Exemplo

Remoção do nodo com chave = 15

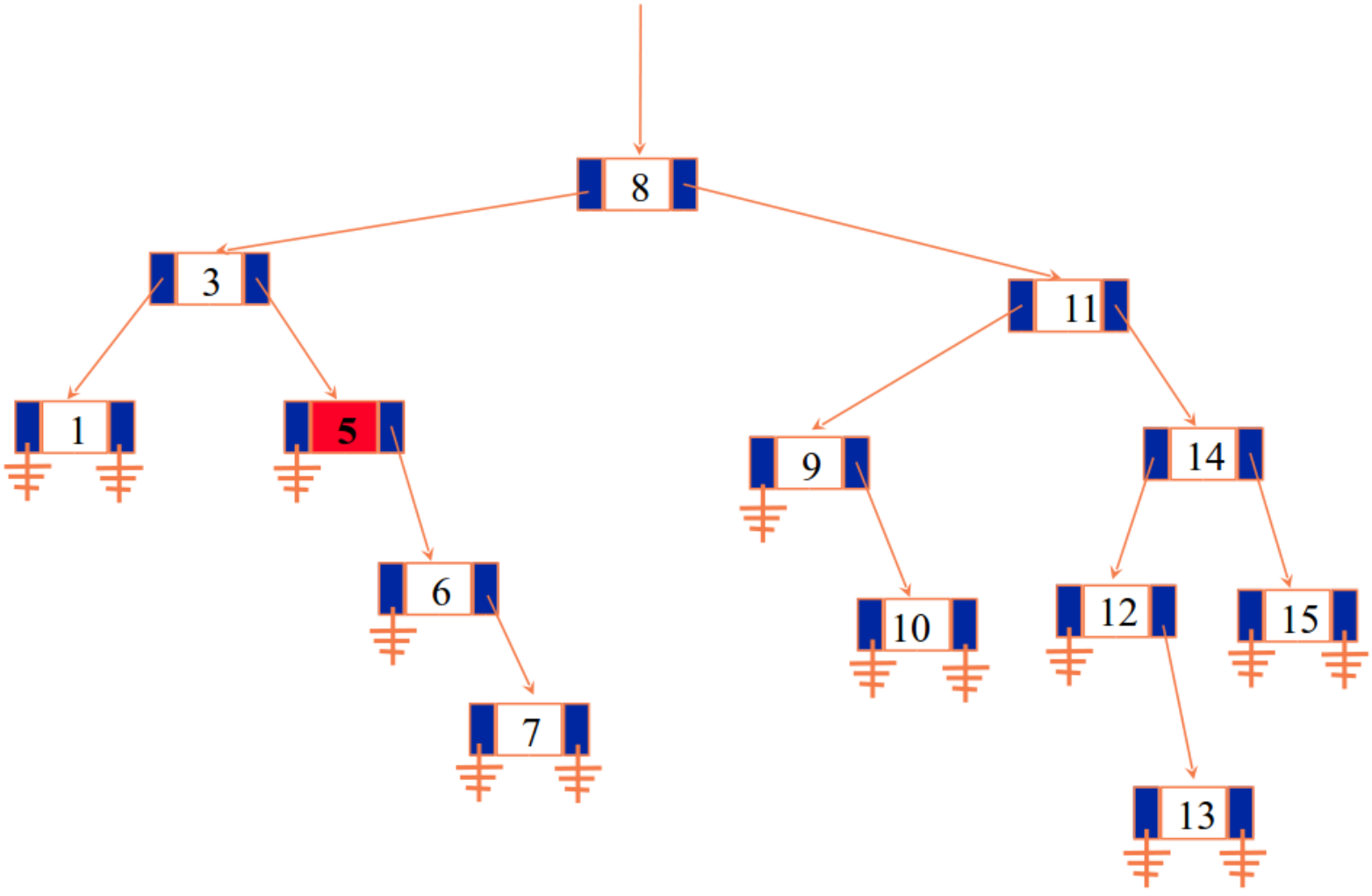


Remoção em uma Arvore de Busca Binária

- 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.

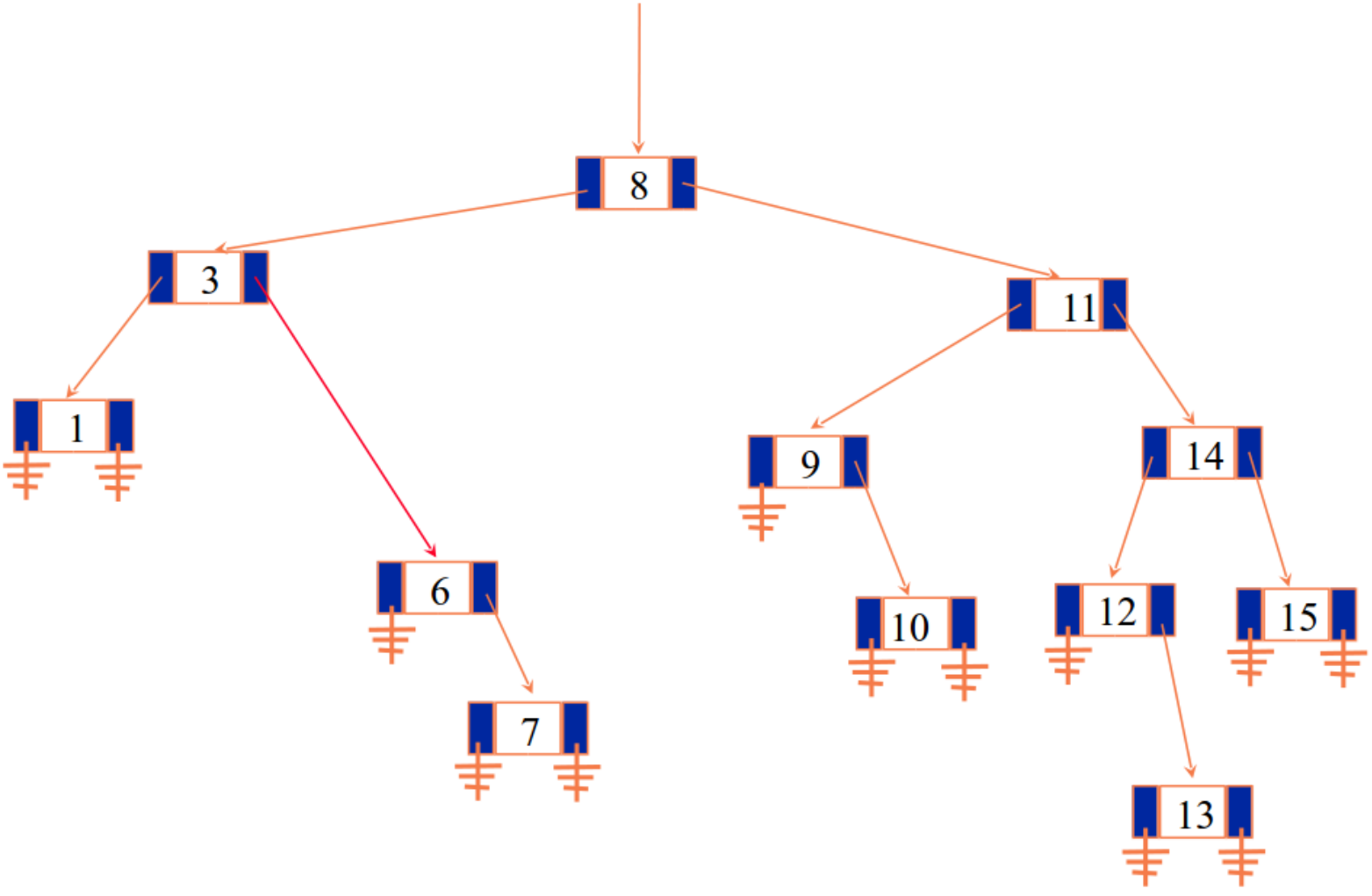
Exemplo

Remoção do nodo com chave = 5



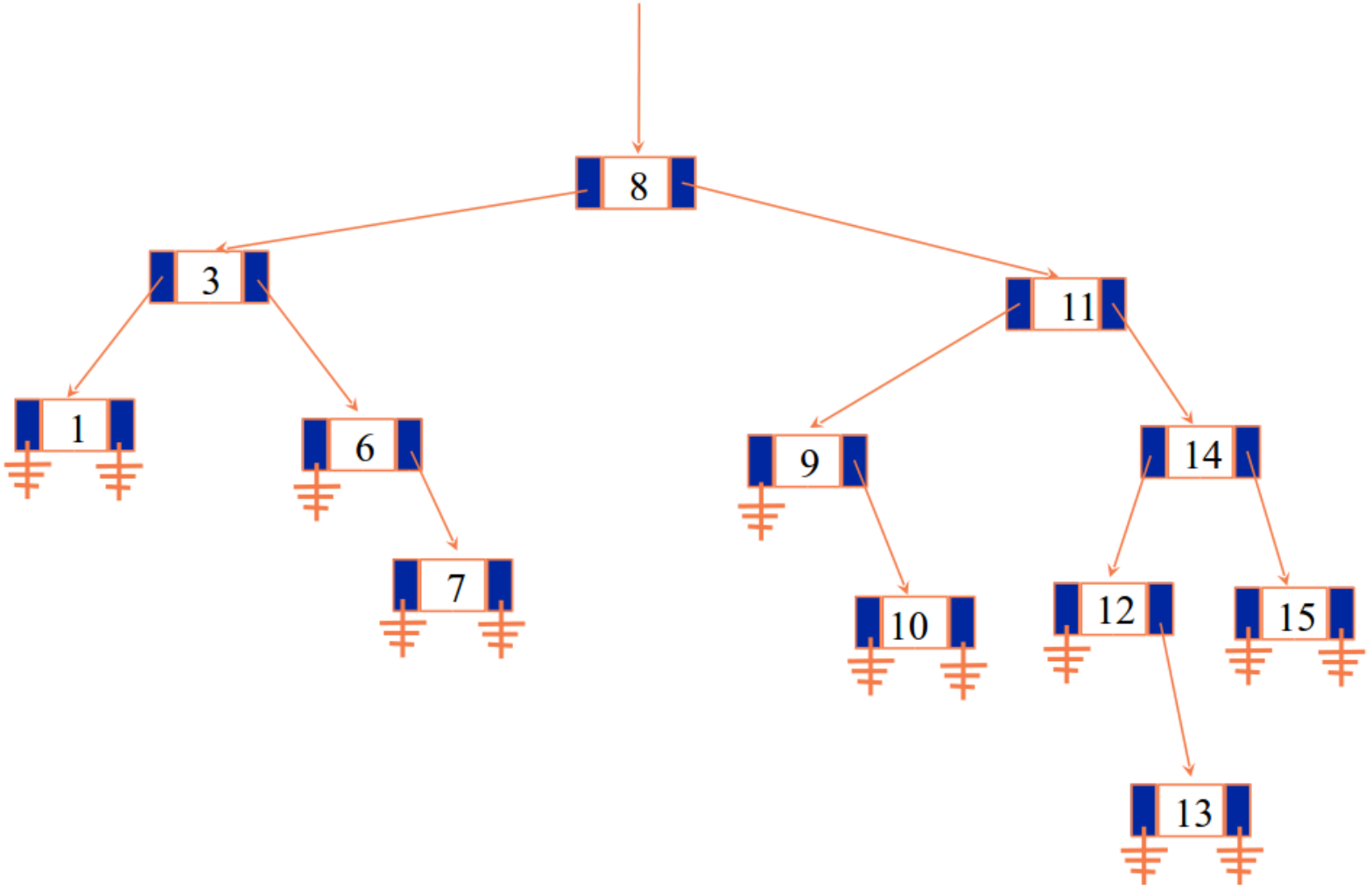
Exemplo

Remoção do nodo com chave = 5



Exemplo

Remoção do nodo com chave = 5

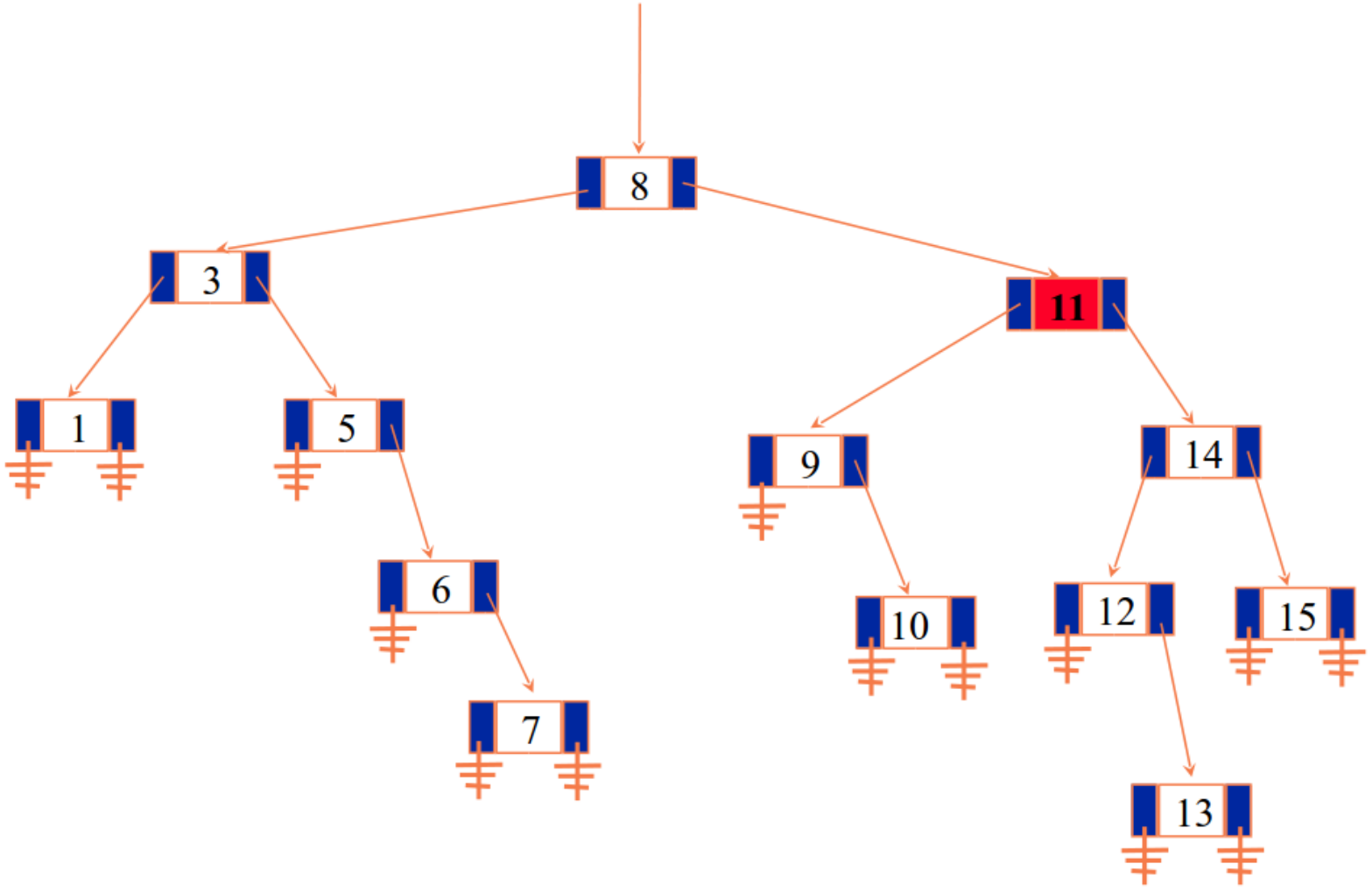


Remoção em uma Arvore de Busca Binária

- 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.

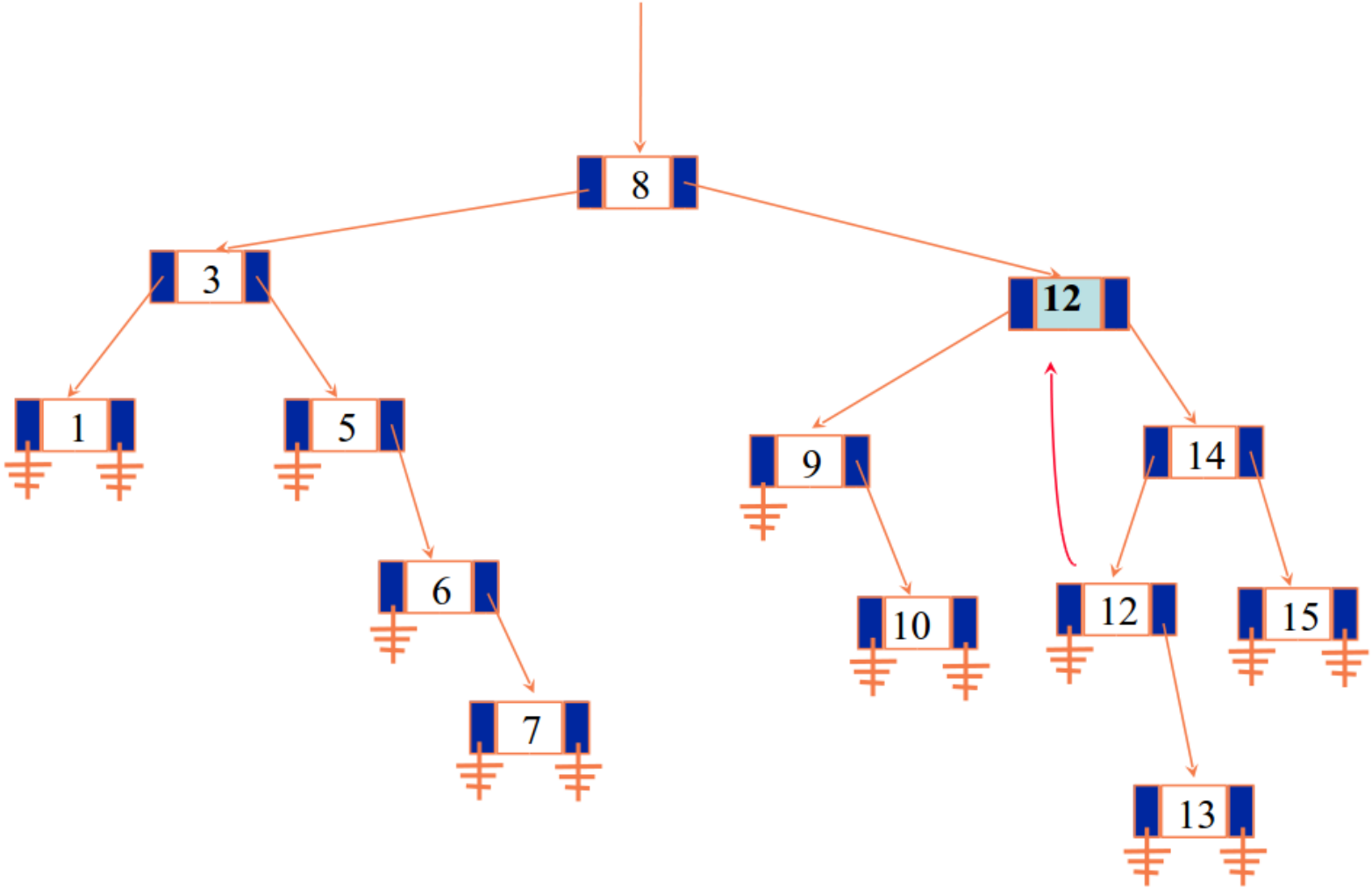
Exemplo

Remoção do nodo com chave = 11



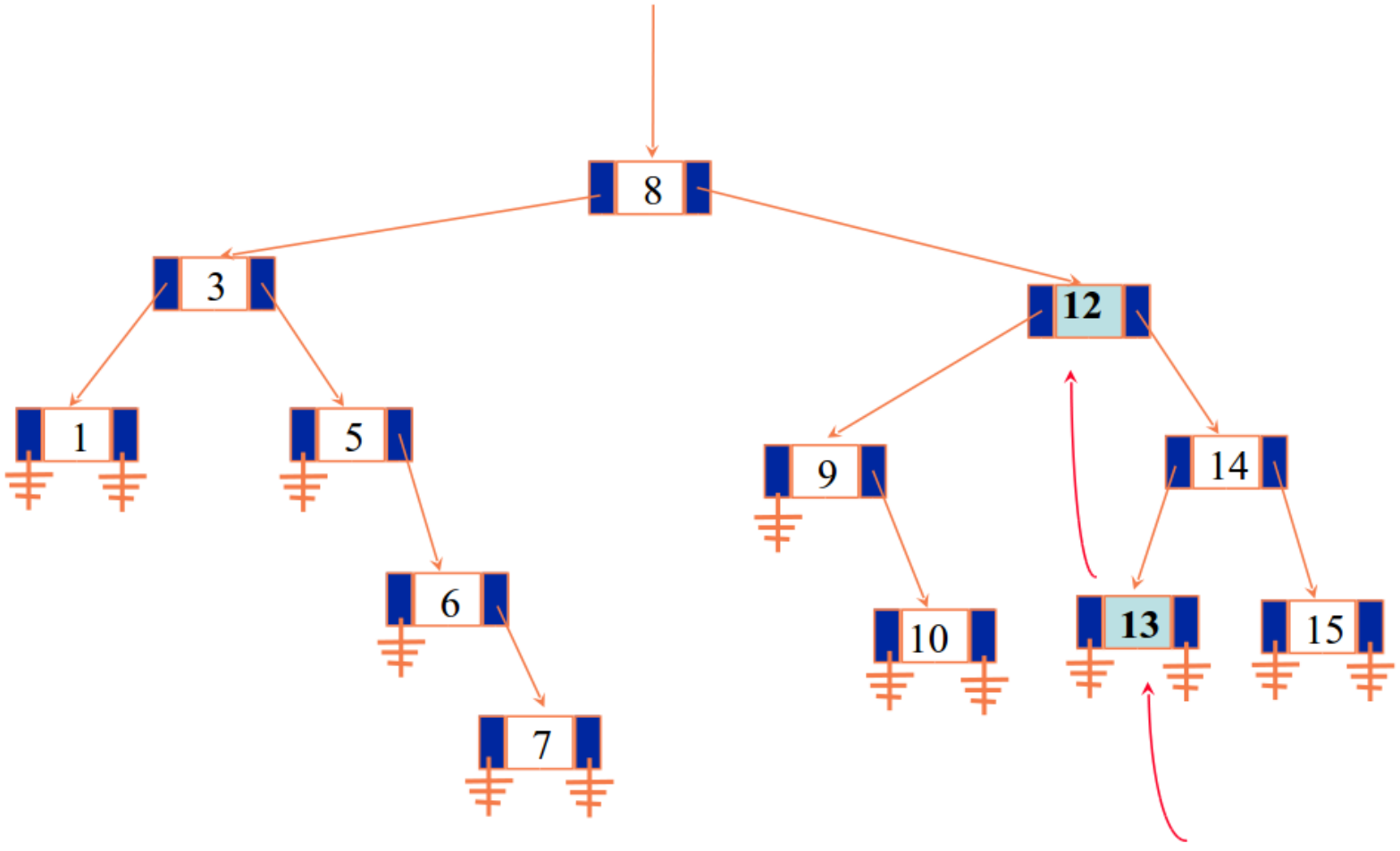
Exemplo

Remoção do nodo com chave = 5



Exemplo

Remoção do nodo com chave = 5



Algoritmo de Remoção

```
tNodo* delete(tNodo *arv, tInfo info)
 tNodo *tmp, *filho;
 inicio
  se (arv = NULO) entao retorne arv
  senao
   se (info < arv->info) // va' 'a esquerda.
    arv->filhoAEsquerda ← delete(info, arv->filhoAEsquerda);
    retorne arv;
   senao
    se (info > arv->info) // va' 'a direita.
     arv->filhoADireita ← delete(info, arv->filhoADireita);
     retorne arv;
    senao // Encontrei elemento que quero deletar.
     se (arv->filhoADireita≠NULO E arv->filhoAEsquerda≠NULO)
         //2 filhos.
      tmp ← minimo(arv->filhoADireita); arv->info ← tmp->
          info;
      arv->filhoADireita ←delete(arv->info,arv->
          filhoADireita);
      retorne arv;
      //(CONTINUA NO PROX SLIDE)
```

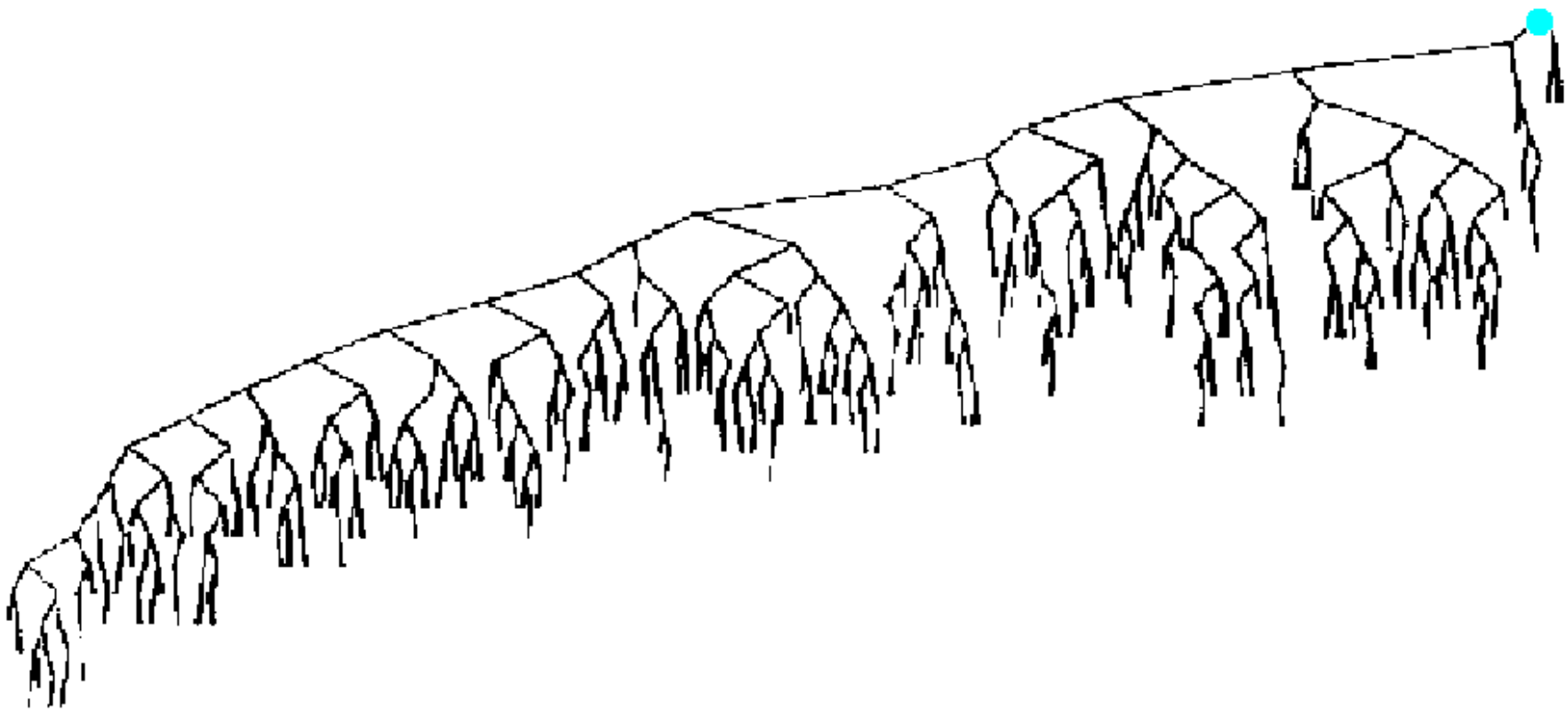
Algoritmo de Remoção

```
      senao  // 1 filho.
       tmp ← arv;
       se (arv->filhoADireita ≠ NULO) entao  // filho 'a
           direita
       filho ← arv->filhoADireita; retorne filho;
      senao
       se (arv->filhoAEsquerda ≠ NULO) entao  // filho 'a
           esquerda
       filho ← arv->filhoAEsquerda; retorne filho;
      senao  // folha
       libere arv;
       retorne NULO;
      fim se
      fim se
      fim se
     fim se
    fim se
   fim se
fim
```

Problemas com Arvores de Busca Binária

- 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 Arvores de Busca Binária



Trabalho

- 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;
- Entregue até a data definida no moodle.

Perguntas?


ine
Informática
Estatística
CTC • UFSC


ARS ET SCIENTIA
UNIVERSIDADE FEDERAL
DE SANTA CATARINA
UFSC