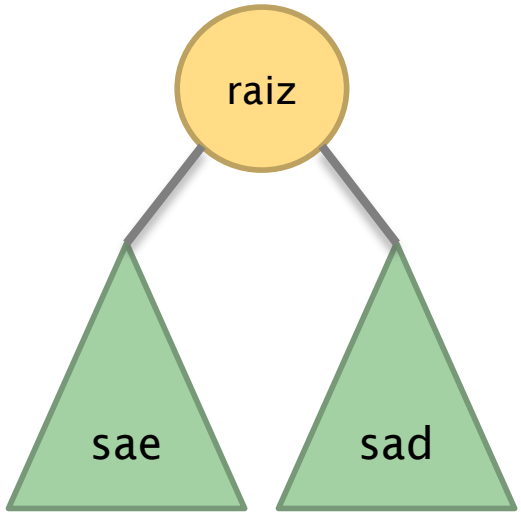
Árvore binária - definição

árvore binária: conjunto finito de nós

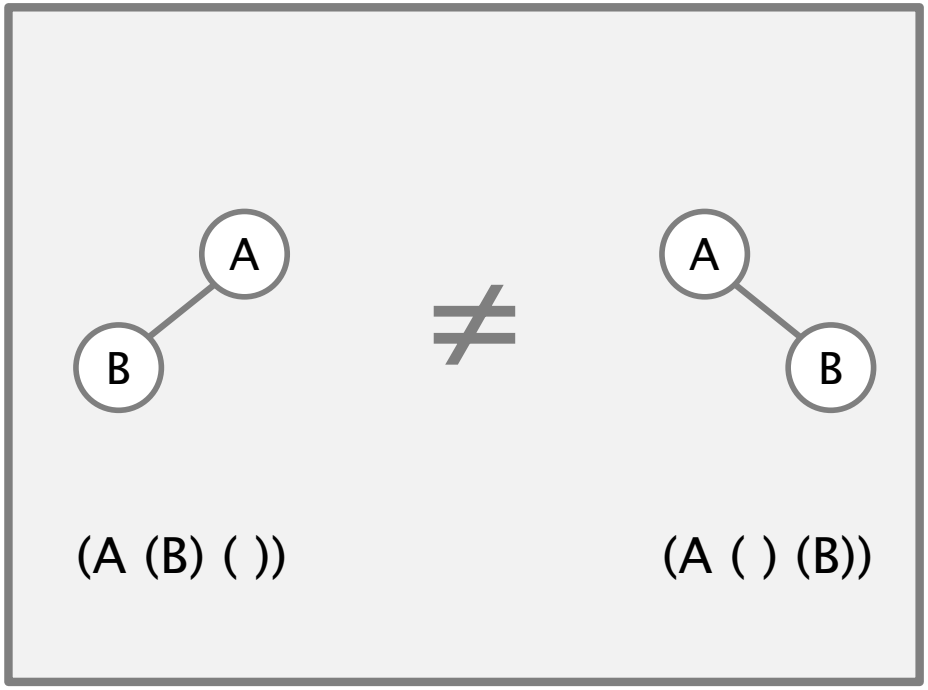
- ∅ (árvore vazia)
- {raiz, sub-árvore esquerda, sub-árvore direita}, onde sae e sad são conjuntos disjuntos

∅ ou




```
/* nó da árvore binária */
struct Node {
    int   raiz;
    Node* sae;
    Node* sad;
};
```

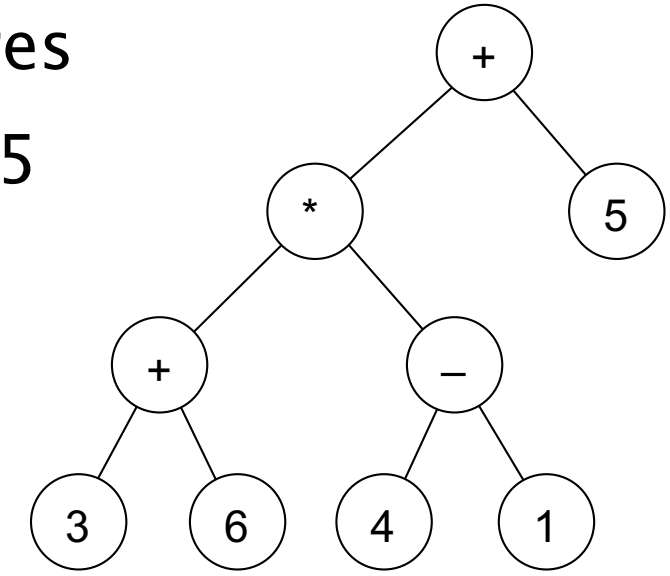
Árvore binária



Exemplo

árvores binárias representando expressões aritméticas:

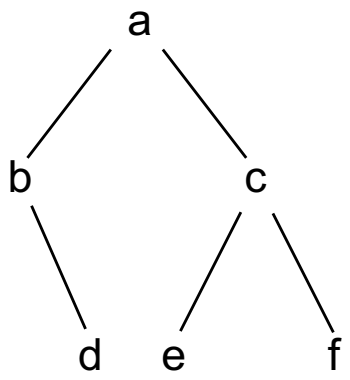
- nós folhas representam operandos
- nós internos operadores
- exemplo: (3+6)*(4-1)+5



Árvores binárias

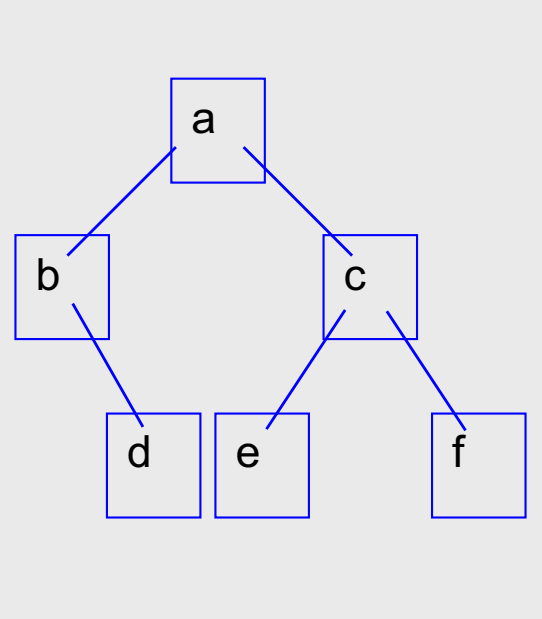
- Notação textual:
  - a árvore vazia é representada por <>
  - árvores não vazias por <raiz sae sad>
  - exemplo:

<a <b <> <d<><><>> > <c <e<><><>> <f<><><>>> >

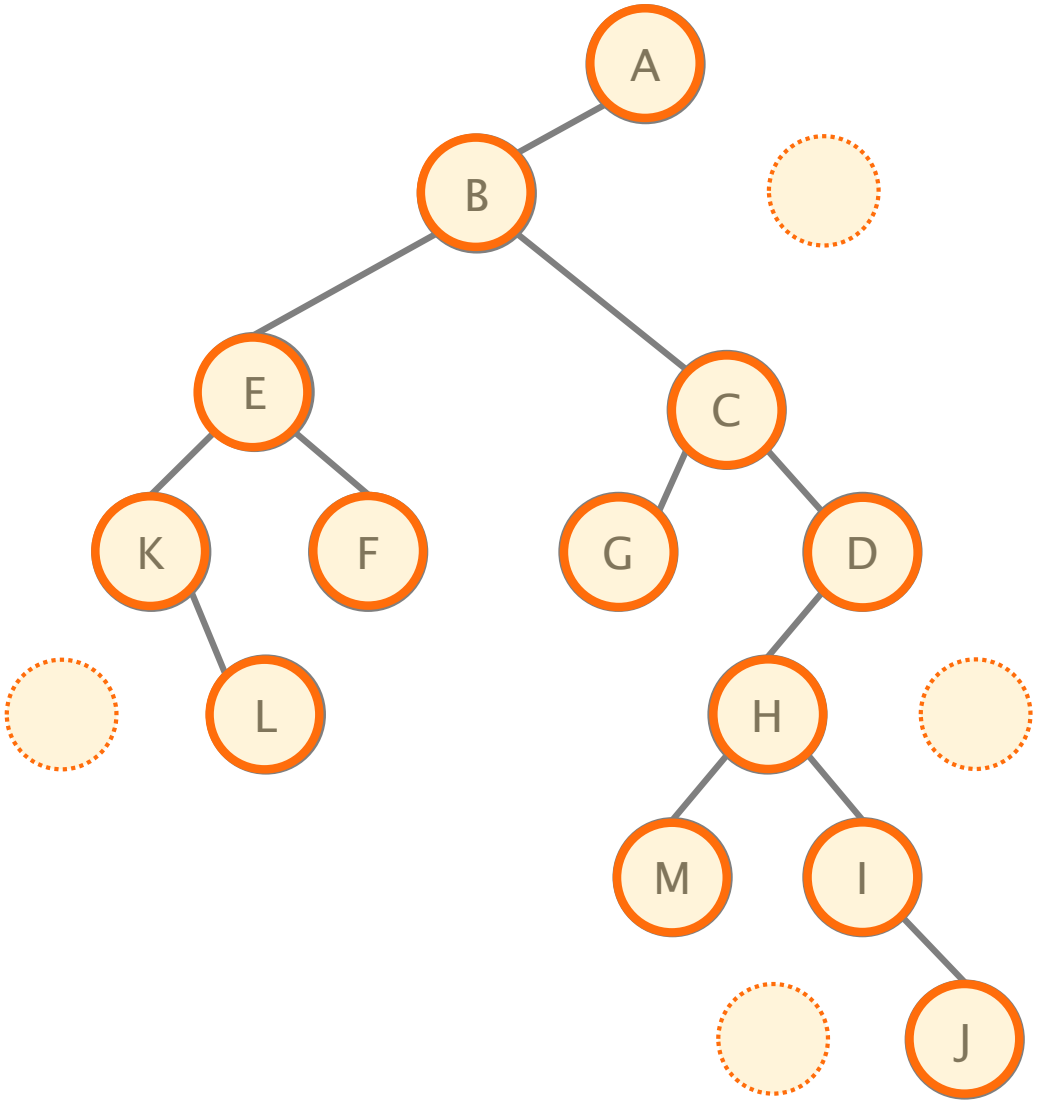


Árvores binárias - Ordens de percurso

- Ordens de percurso:
  - *pré-ordem*:
    - trata *raiz*, percorre *sae*, percorre *sad*
    - exemplo: a b d c e f
  - *ordem simétrica*:
    - percorre *sae*, trata *raiz*, percorre *sad*
    - exemplo: b d a e c f
  - *pós-ordem*:
    - percorre *sae*, percorre *sad*, trata *raiz*
    - exemplo: d b e f c a

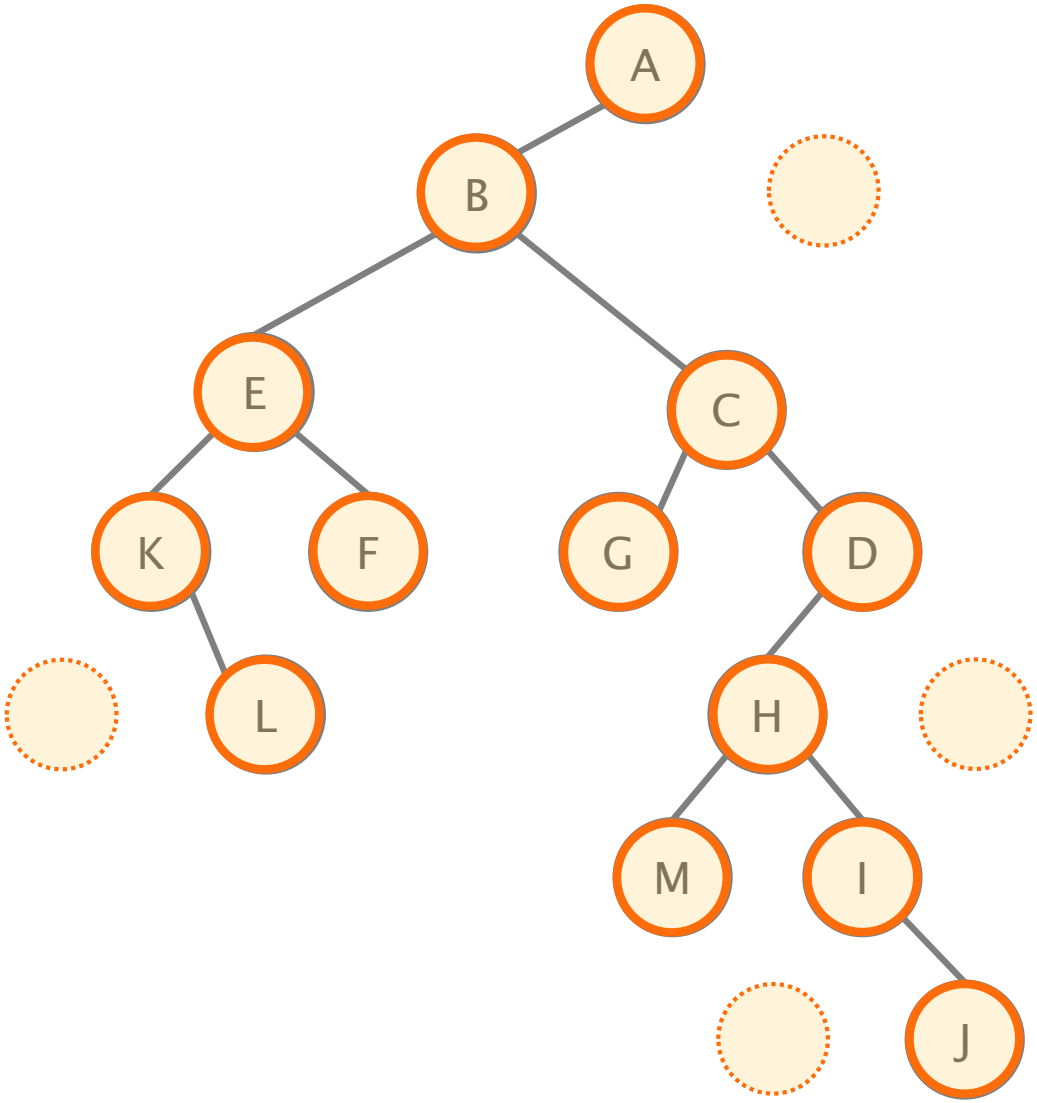


Percurso – pré-ordem



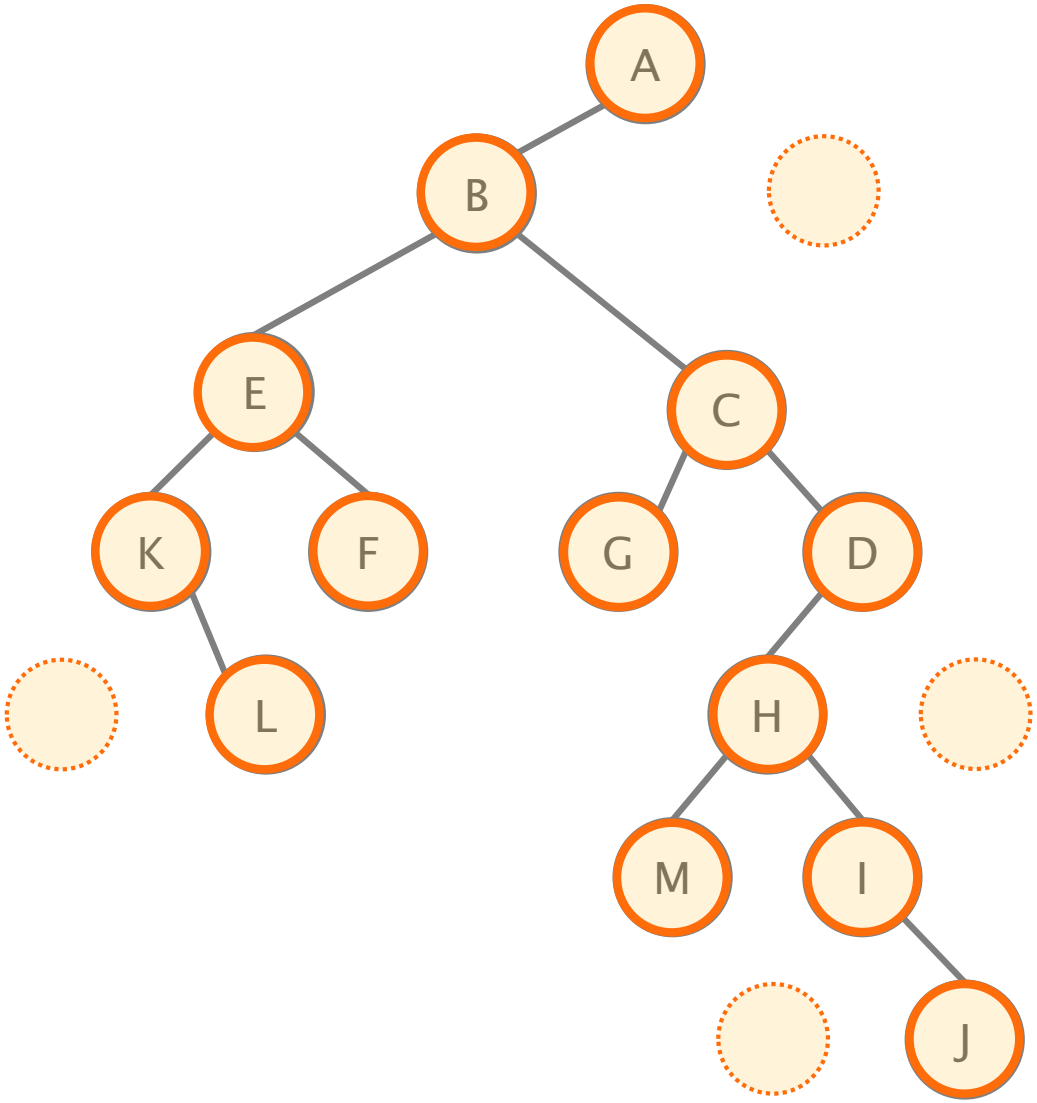
A	B	E	K	*	L	F	C	G	D	H	M	I	*	J	*	*

Percurso – pós-ordem



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

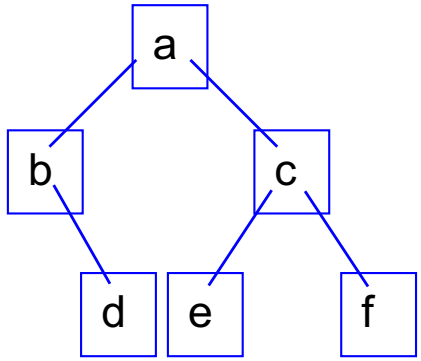
Percurso – ordem simétrica



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

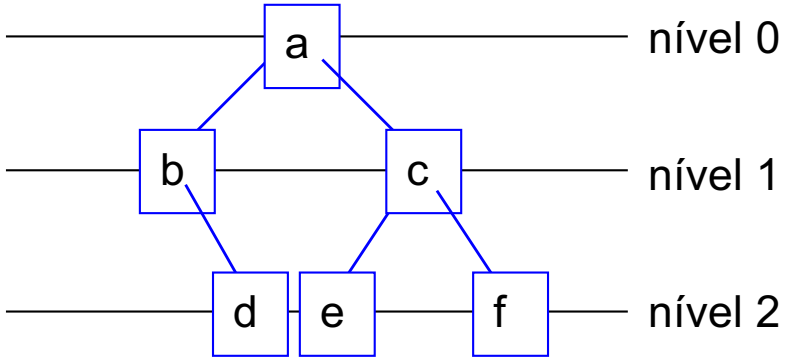
Árvores binárias - Altura

- Propriedade fundamental de árvores
  - só existe um caminho da raiz para qualquer nó
- Altura de uma árvore
  - 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



Árvores binárias - Altura

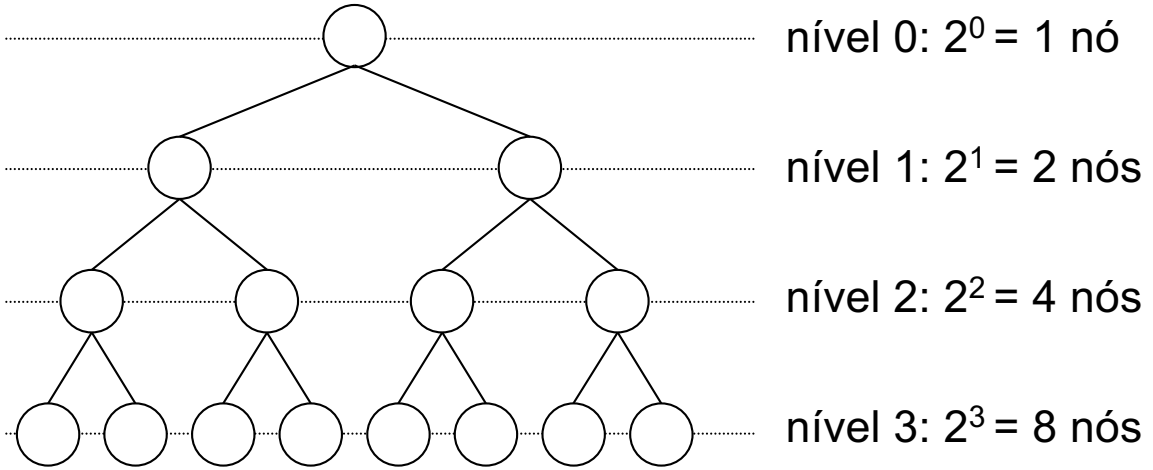
- Nível de um nó
  - a raiz está no nível 0, seus filhos diretos no nível 1, ...
  - o último nível da árvore é a altura da árvore



Árvores binárias - Altura

- Árvore cheia
  - todos os seus nós internos têm duas sub-árvores associadas
  - número *n* de nós de uma árvore cheia de altura *h*

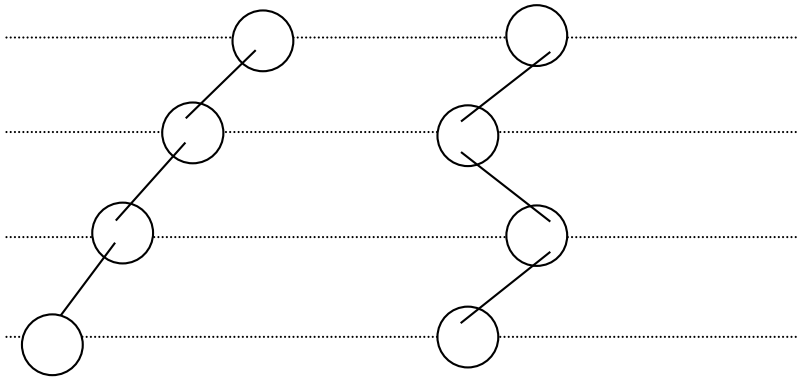
    $n = 2^{h+1} - 1$



Árvores binárias - Altura

- Árvore degenerada
  - todos os seus nós internos têm uma única sub-árvore associada
  - número *n* de nós de uma árvore degenerada de altura *h*

    $n = h+1$



Árvores binárias - Altura

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

Árvore binária - conceitos

número máximo de nós no nível *i*:

2^i

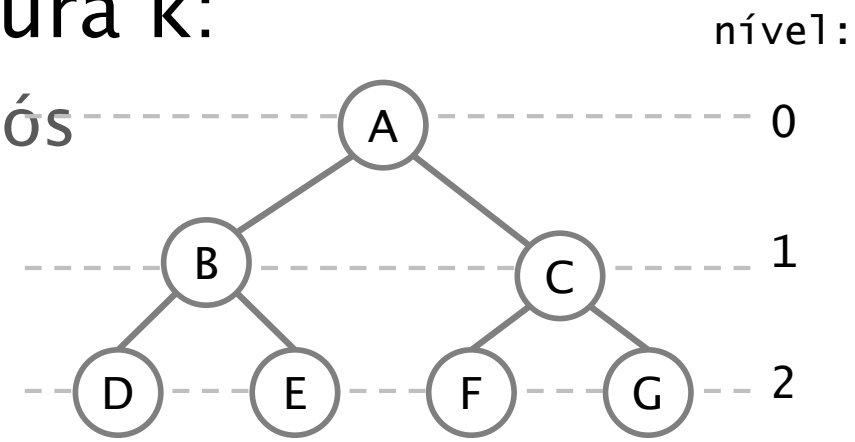
número máximo de nós na árvore de altura *k*:

$2^{k+1} - 1, k \geq 0 \quad (= 2^k \ldots + 2^2 + 2 + 1)$

árvore binária **cheia** de altura k:

árvore que possui $2^{k+1} - 1$ nós

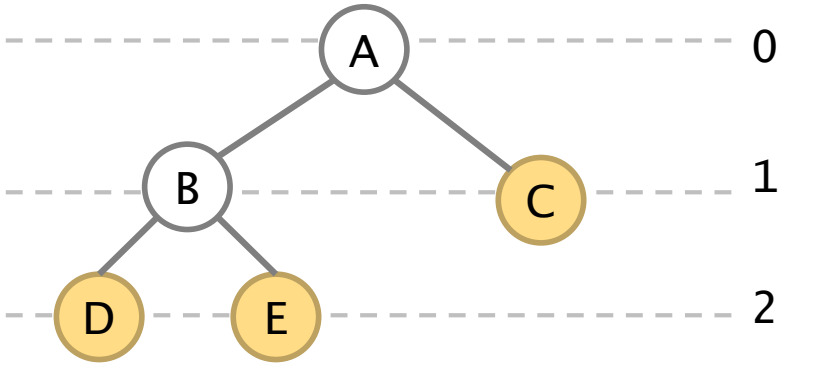
se k = 2, árvore binária cheia possui $2^3-1 = 7$ nós



Árvore binária – conceitos (cont.)

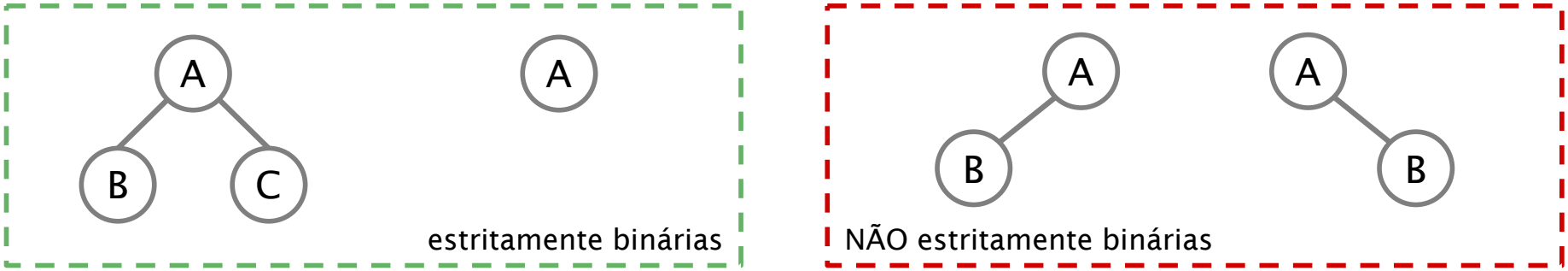
árvore binária **completa**

toda folha está no último ou penúltimo nível



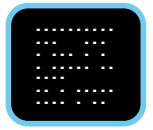
árvore **estritamente binária**

cada nó tem 0 ou dois filhos



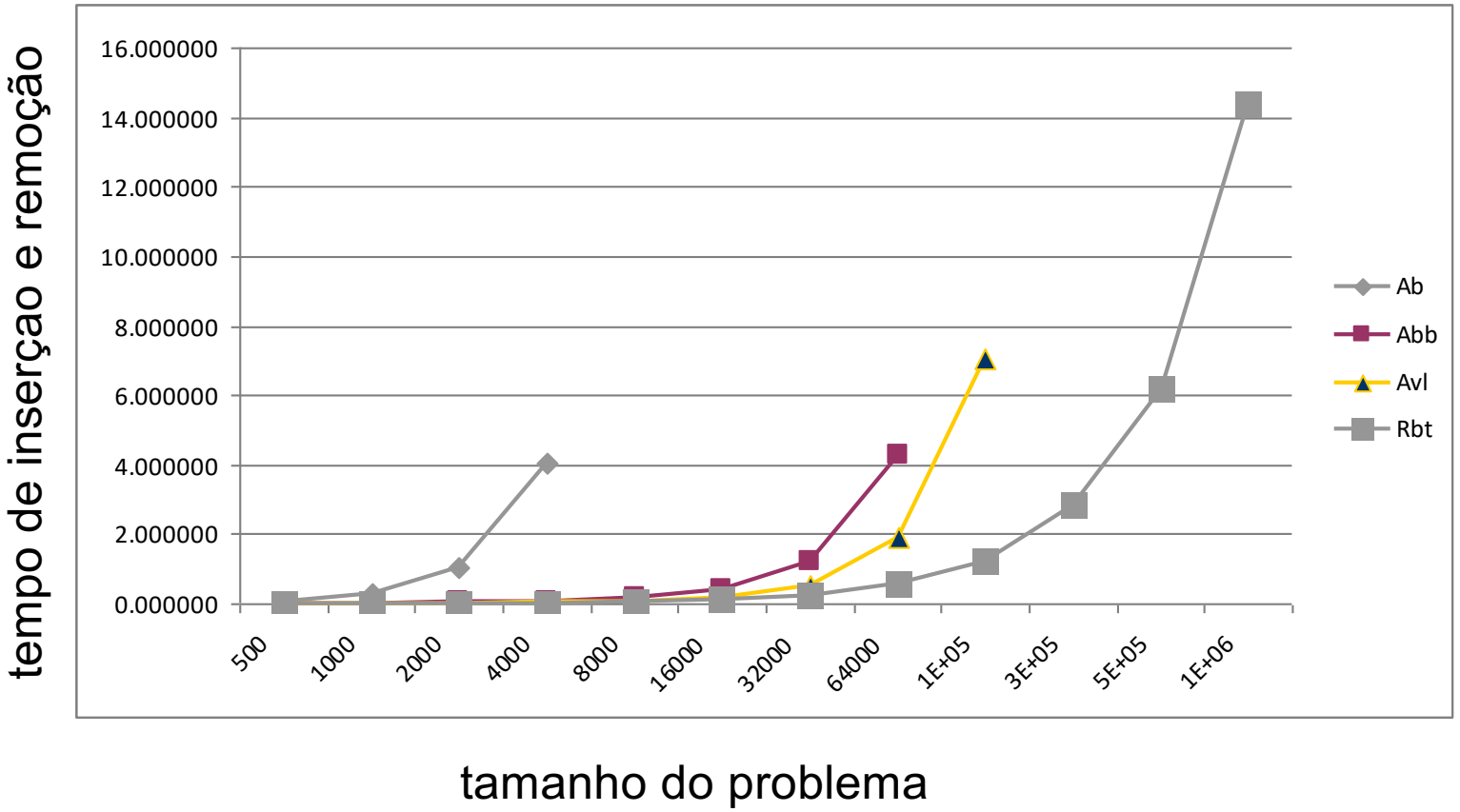
Classe Árvore Binária

```
struct AbbNo
{
    int _chave;
    AbbNo* _esq;
    AbbNo* _dir;
};

class Abb
{
public:
    Abb();
    Abb(c int& info);
    Abb(const Abb& orig);
    ~Abb();
    void insere(int info);
    bool remove(int info);
    void mostra(char* title);
    int  altura();
    int tamanho();

private:
    AbbNo* _raiz;
};
```

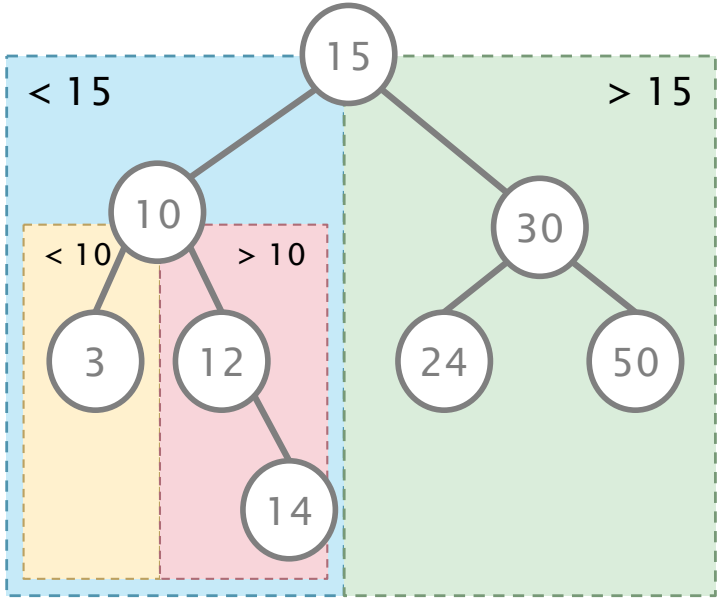


© 2012 DI, PUC-Rio · Estruturas de Dados Avançadas · 2012.1

O que vamos estudar de árvores binárias?



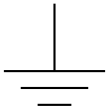
Árvores binárias de busca (Abb)



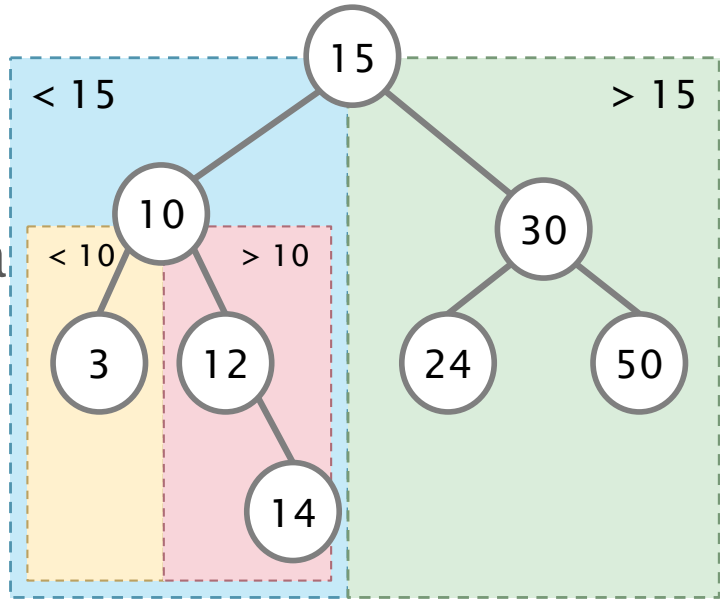
Definição de uma Abb

Abb é uma árvore binária vazia,

ou



1. cada nó possui uma chave
2. as chaves na sub-árvore esquerda (se houver) são menores do que a chave da raiz
3. as chaves na sub-árvore direita (se houver) são maiores do que a chave da raiz
4. as sub-árvores esquerda e direita são árvores binárias de busca



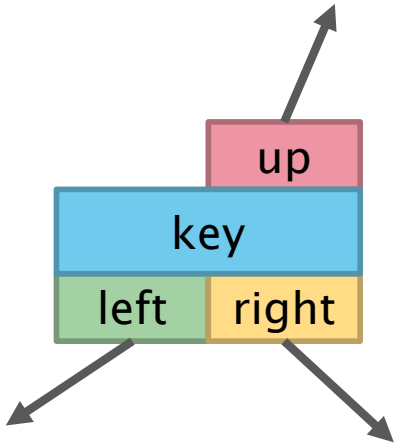
Uma implementação de Abb em C++

abb.hpp

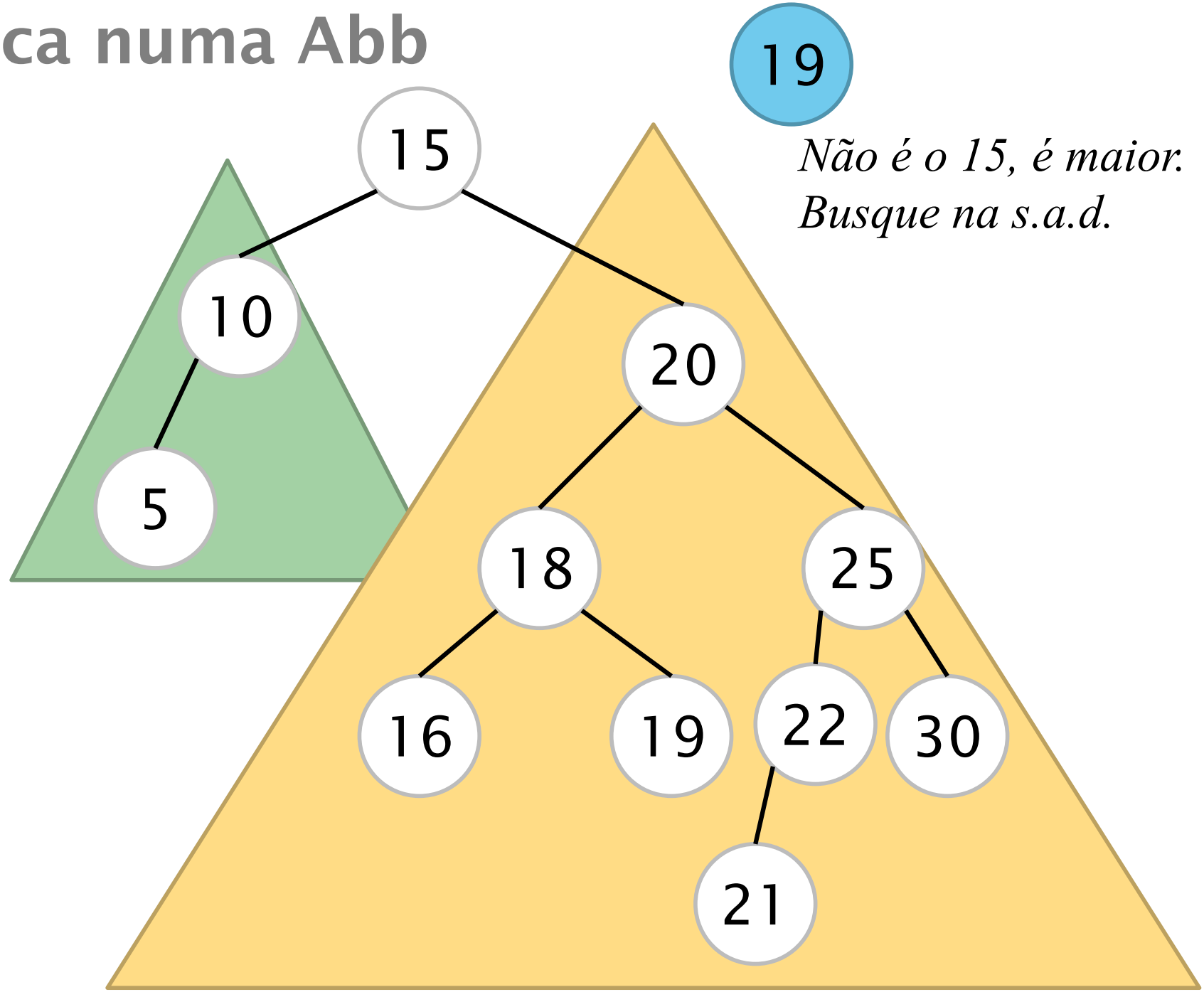
```
struct Node {
    int   _key;
    Node* _up;
    Node* _left;
    Node* _right;
};

class Abb {
  public:
    Abb();
    Abb(int key);
    …


  private:
      Node* _root;
 };
```

abb.cpp

```
Abb::Abb() {
    _root = nullptr;
}

Abb::Abb(int key) {
    _root = new Node;
    _root->_key = key;
    _root->_left = nullptr;
    _root->_right = nullptr;
    _root->_up = nullptr;
}
```



Busca numa Abb



Não é o 15, é maior.
Busque na s.a.d.

Algoritmo para busca em ABBs

```
Node* Abb::search(int key);
```

1. Começe a busca pelo nó raiz
2. Se a árvore for **vazia** retorne **NULL**
3. CC se a chave procurada for **menor** que a chave do nó, **procure na sub-árvore à esquerda** e responda com a resposta que você receber
4. CC se a chave procurada for **maior** que a chave do nó, **procure na sub-árvore à direita** e responda com a resposta que você receber
5. CC se for **igual** responda com o **endereço do nó**

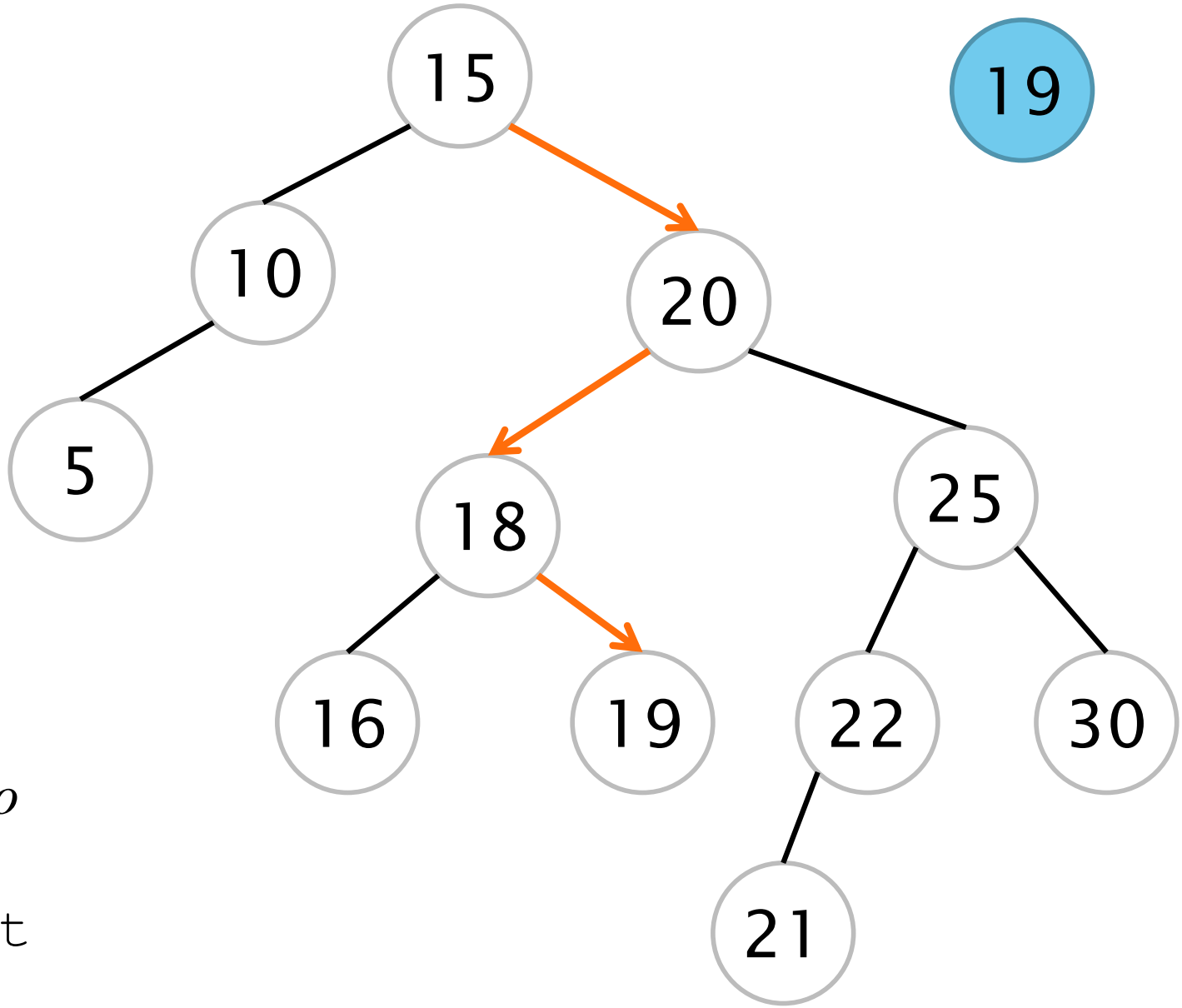
CC = caso contrário

Busca numa abb

1. Começe a busca pelo nó raiz
2. Se a árvore for **vazia** retorne **NULL**
3. CC se a chave procurada for **menor** que a chave do nó, **procure na sub-árvore à esquerda** e responda com a resposta que você receber
4. CC se a chave procurada for **maior** que a chave do nó, **procure na sub-árvore à direita** e responda com a resposta que você receber
5. CC se for **igual** responda com o **endereço do nó**

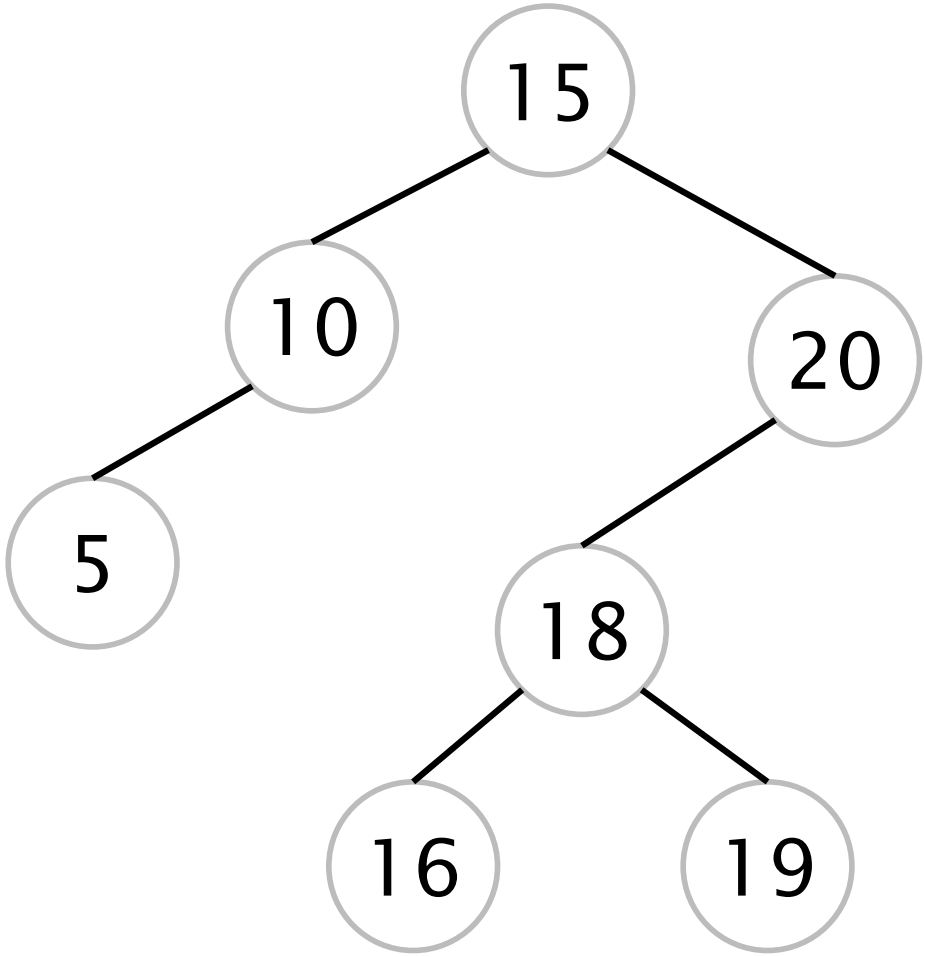
```
Node* abb::search(int key) {
    return searchrec(_root, key);
}



Node* Abb::searchrec(Node* node, int key) {
    if (node == nullptr) return nullptr;
    else if (node->_key == key) return node;
    else if (key > node->_key) return searchrec(node->_right, key);
    else return searchrec(node->_left, key);
}
```

Busca iterativa numa Abb

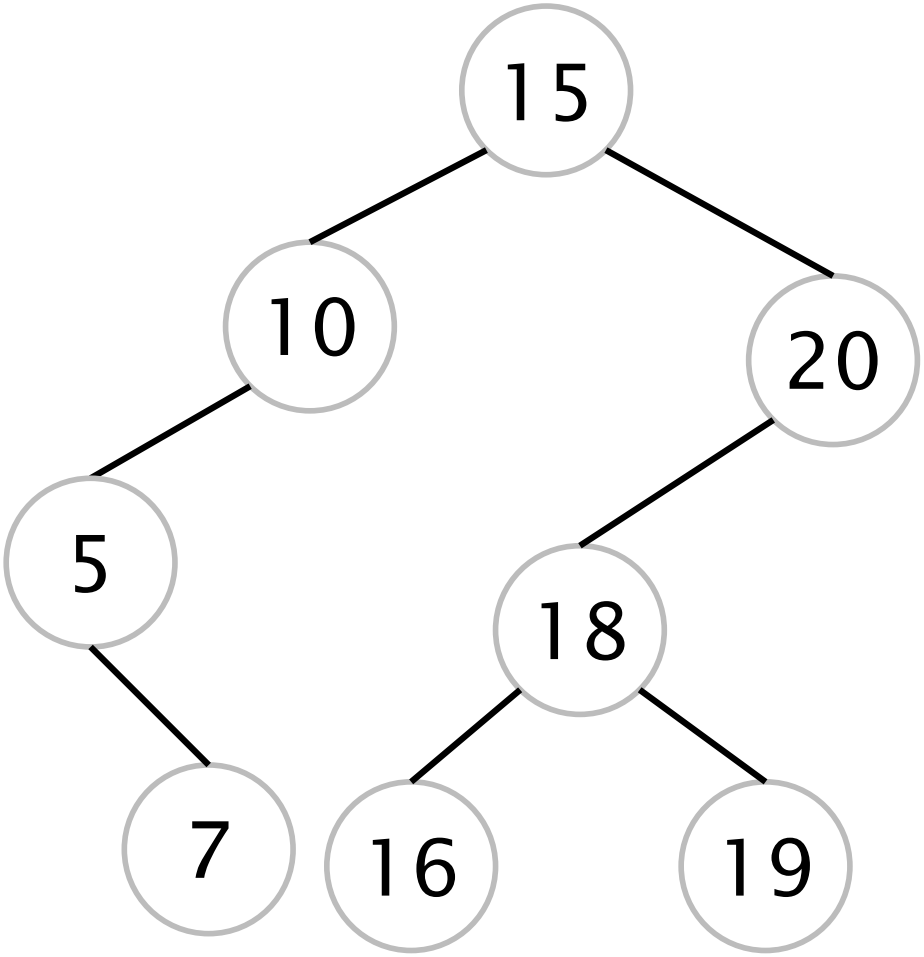


Busca como em lista, com o `next` *variando*

Menor nó numa Abb



Menor nó numa Abb



Menor nó numa Abb

```
int Abb::min ( );
```

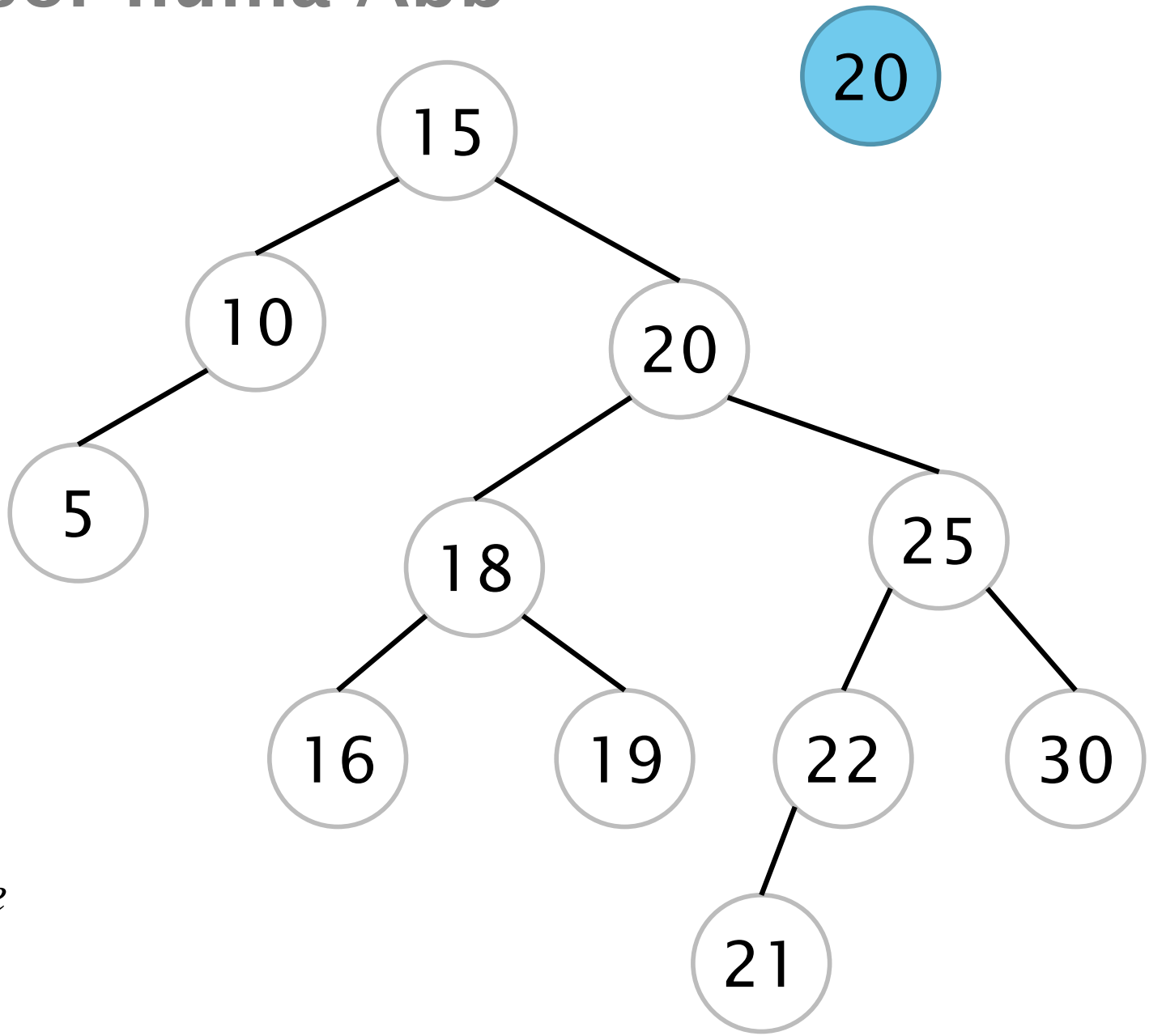
É a chave do nó mais à esquerda da árvore

1. Começando pelo nó raiz
2. Se a árvore for vazia retorne -1
3. Caso contrário, caminhe sempre à esquerda enquanto o filho esquerdo não for NULL

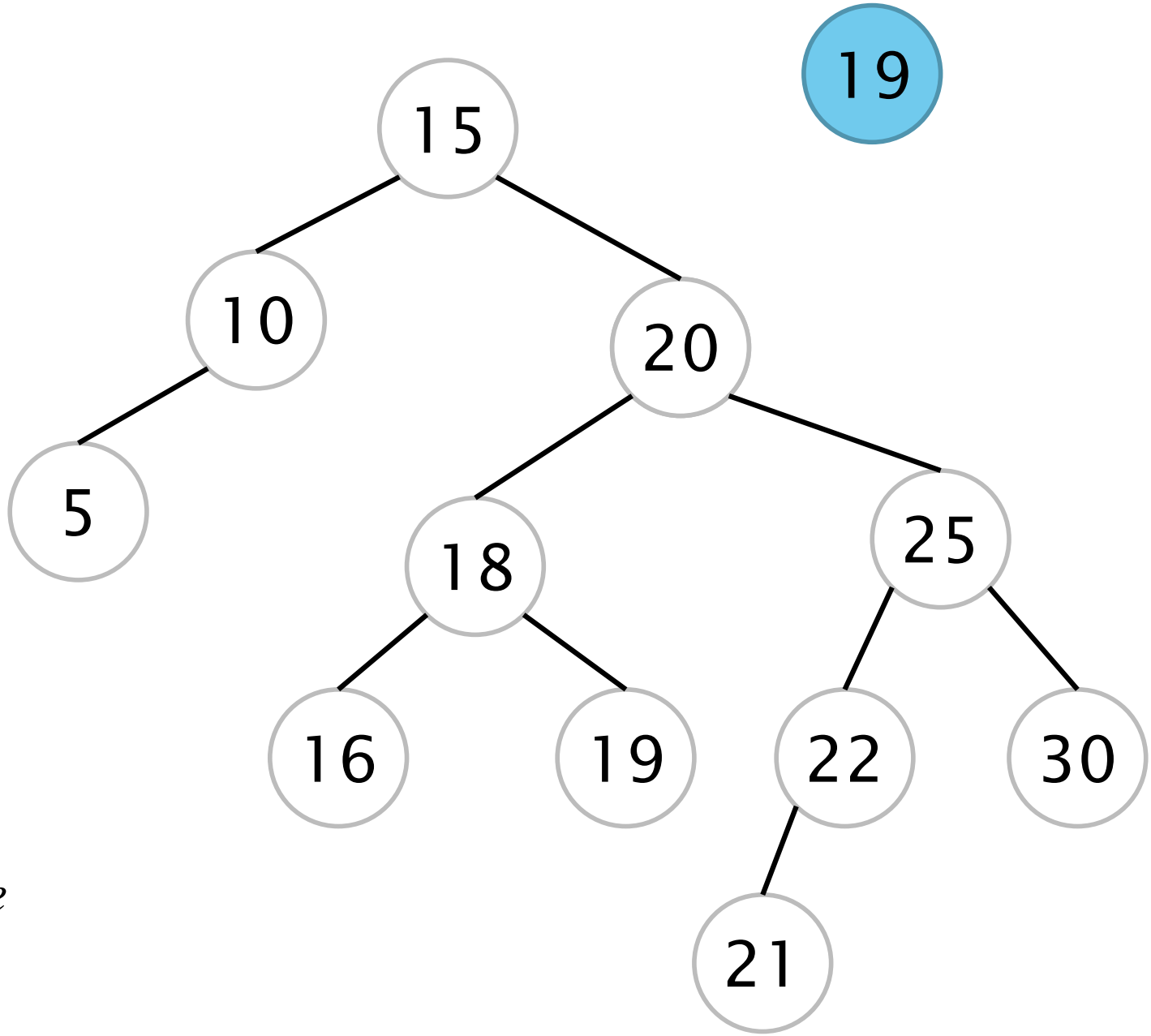
Menor chave numa Abb

```
int Abb::min(){
    Node* p=_root;
    if (p==nullptr) return -1;
    while (p->_left!=nullptr) p = p->_left;
    return p->_key;
}
```

Sucessor numa Abb



Quem é o sucessor de 20?



Quem é o sucessor de 19?

Sucessor numa Abb (next)

```
int Abb::next (int key);
```

1. Ache o o nó que contém a a key;
2. Se não achar return NULL
3. CC se o nó tiver uma sub-árvore à direita, retorne o mínimo dela
4. CC suba na árvore procurando o primeiro ancestral que seja maior que seu filho. Ou seja, o nó corrente vai estar na sua sub-árvore à esquerda. Se nessa busca você chegar na raiz (ancestral NULL), retorne NULL.

CC = caso contrário

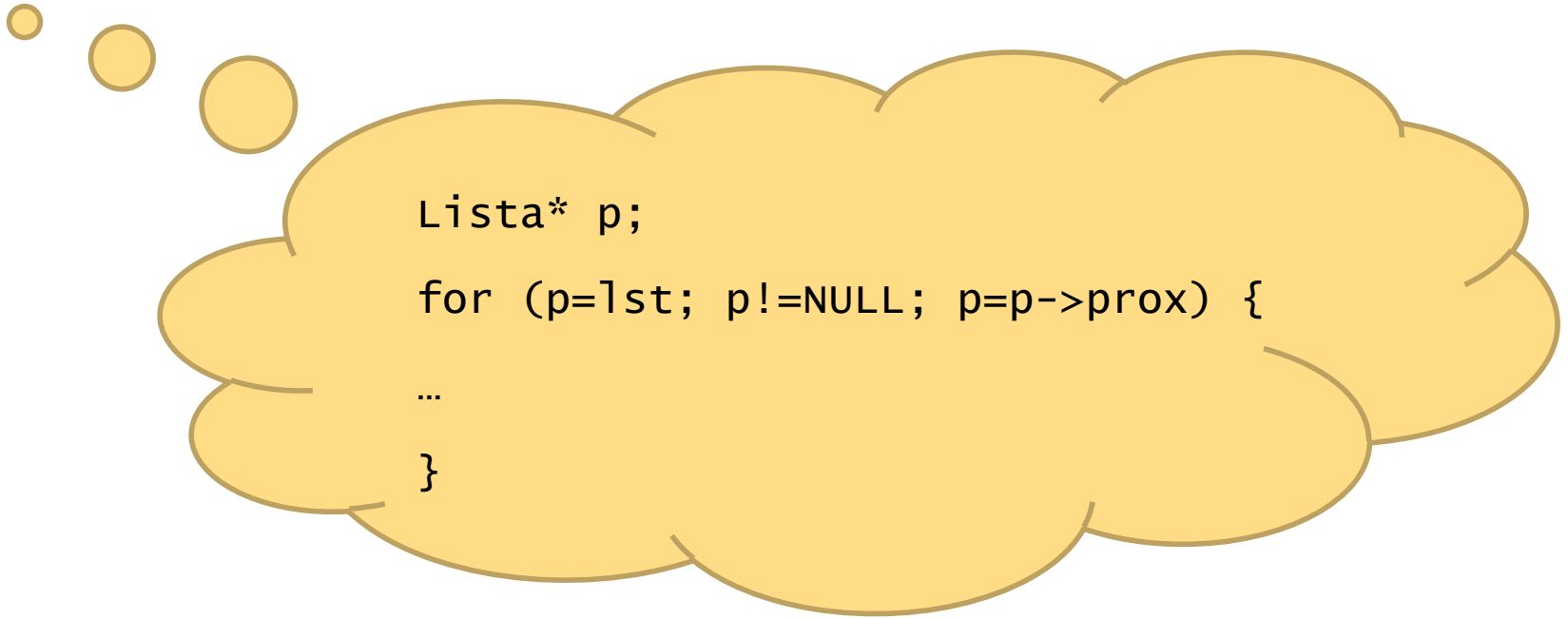
Sucessor numa Abb (prox)

```
bool Abb::next() {
    if (_cursor == nullptr) return false; // already the last
    if (_cursor->_right != nullptr) { // has a right sub tree
        _cursor = _cursor->_right;
        while (_cursor->_left != nullptr)
            _cursor = _cursor->_left;
        return true;
    } else {
        while (_cursor != nullptr) {
            if (_cursor->_up != nullptr && _cursor->_up->_key > _cursor->_key) {
                _cursor = _cursor->_up;
                return true;
            }
            _cursor = _cursor->_up;
        }
        return false;
    }
}
```

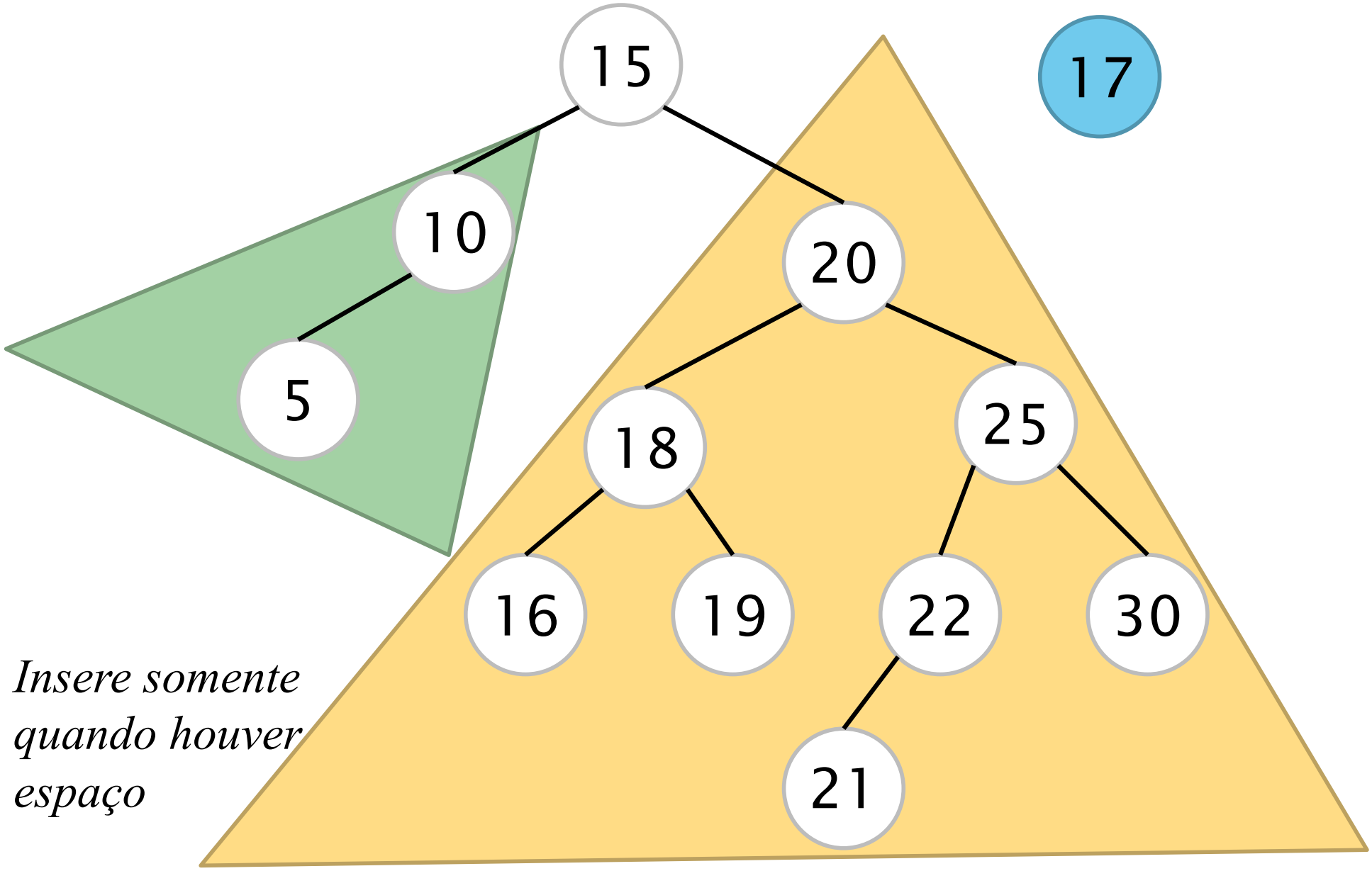
```
private:
    Node* _root;
    Node* _cursor;

};
```

Percorrendo em ordem os nós de uma Abb

```
a.first();
cout<<"Order first to last: "<<a.value() << ", ";
while(a.next()) {
    cout << a.value() << ", ";
}
cout<<endl;
```

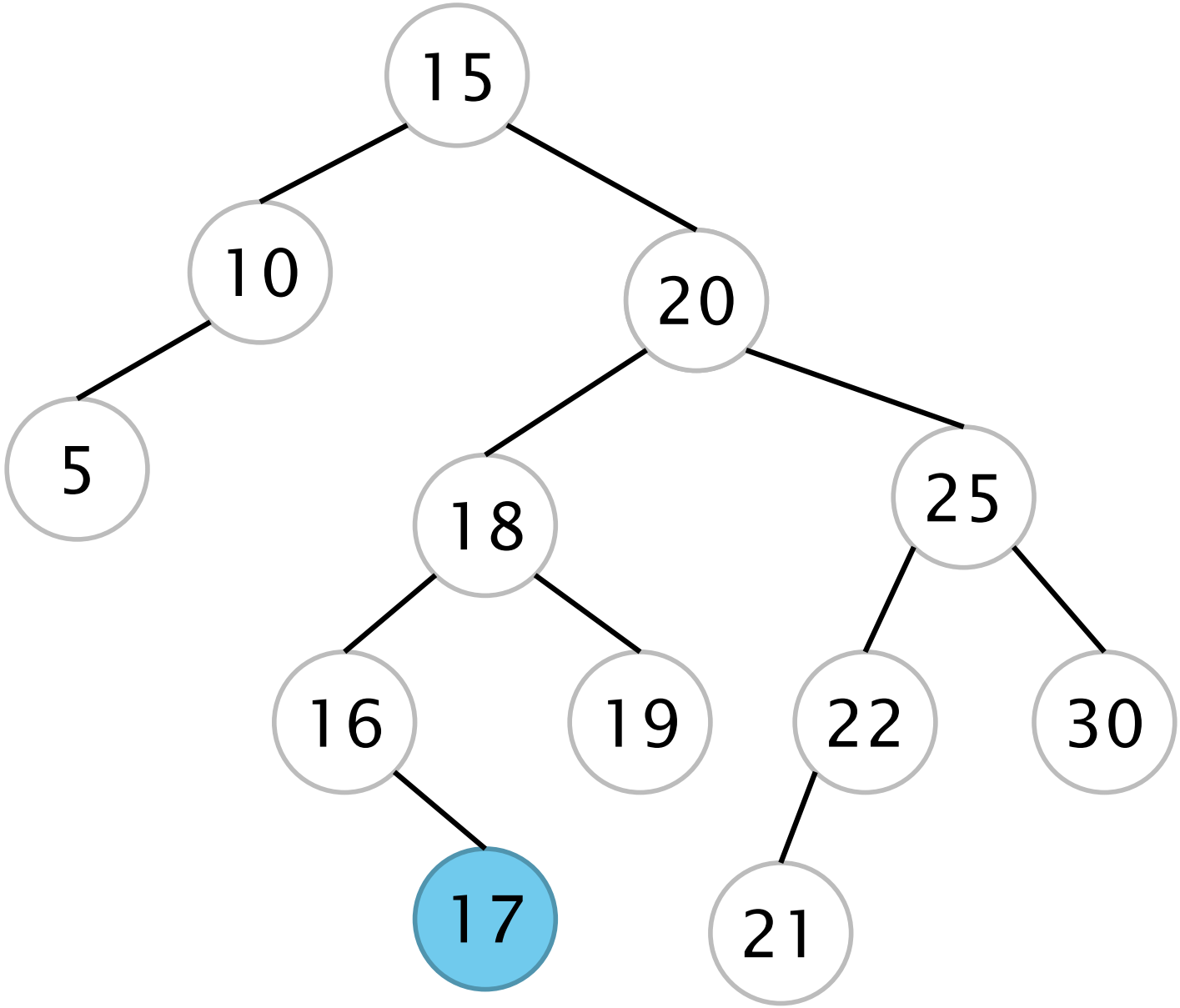


Inserção numa Abb



Insere somente quando houver espaço

Inserção numa Abb



Inserção numa abb

```
Abb* abb_insere (Abb* r, int val);
```

1. Começando pelo nó raiz
2. Se o nó for NULL crie num nó com a chave dada e retorne o endereço dele.
3. CC se a chave dada for **maior** que a chave corrente:
   1. o nó tiver uma sub-árvore à **direita**, insira nela a chave
   2. CC crie um nó com a chave dada e este será o filho à direita do nó corrente
4. CC se a chave dada for **menor** que a chave corrente:
   1. o nó tiver uma sub-árvore à **esquerda**, insira nela a chave
   2. CC crie um nó com a chave dada e este será o filho à esquerda do nó corrente
5. CC se a chave for **igual**, troque a informação associada à chave (na árvore de inteiros não faça nada)

Inserção recursiva numa abb

```
void Abb::insert(int key) {
    _root = insertrec(_root, key);
    _root->_up = nullptr;
}
```

```
Node* Abb::insertrec(Node* a, int key) {
    if (a == nullptr) {
        a = new Node;
        a->_key = key;
        a->_left = a->_right = a->_up = nullptr;
    }









    }
    return a;
}
```

Inserção iterativa numa abb

```
Abb* abb_insere_iterativa (Abb* r, int val);
```

1. Se a árvore for vazia crie um nó e retorne
2. CC começe a busca pelo nó raiz, desça na árvore mantendo o nó anterior (pai)
3. Enquanto o nó não for **NULL** ou não contiver a chave dada:
    1. Se a chave do nó for **maior** que a chave dada, vá para o filho à **direita**
    2. Se a chave do nó for **menor** que a chave dada, vá para o filho à **esquerda**
4. Crie o nó com a chave dada e o coloque como filho do pai

Inserção recursiva numa abb

```
void Abb::insert(int key) {
    _root = insertrec(_root, key);
    _root->_up = nullptr;
}
```

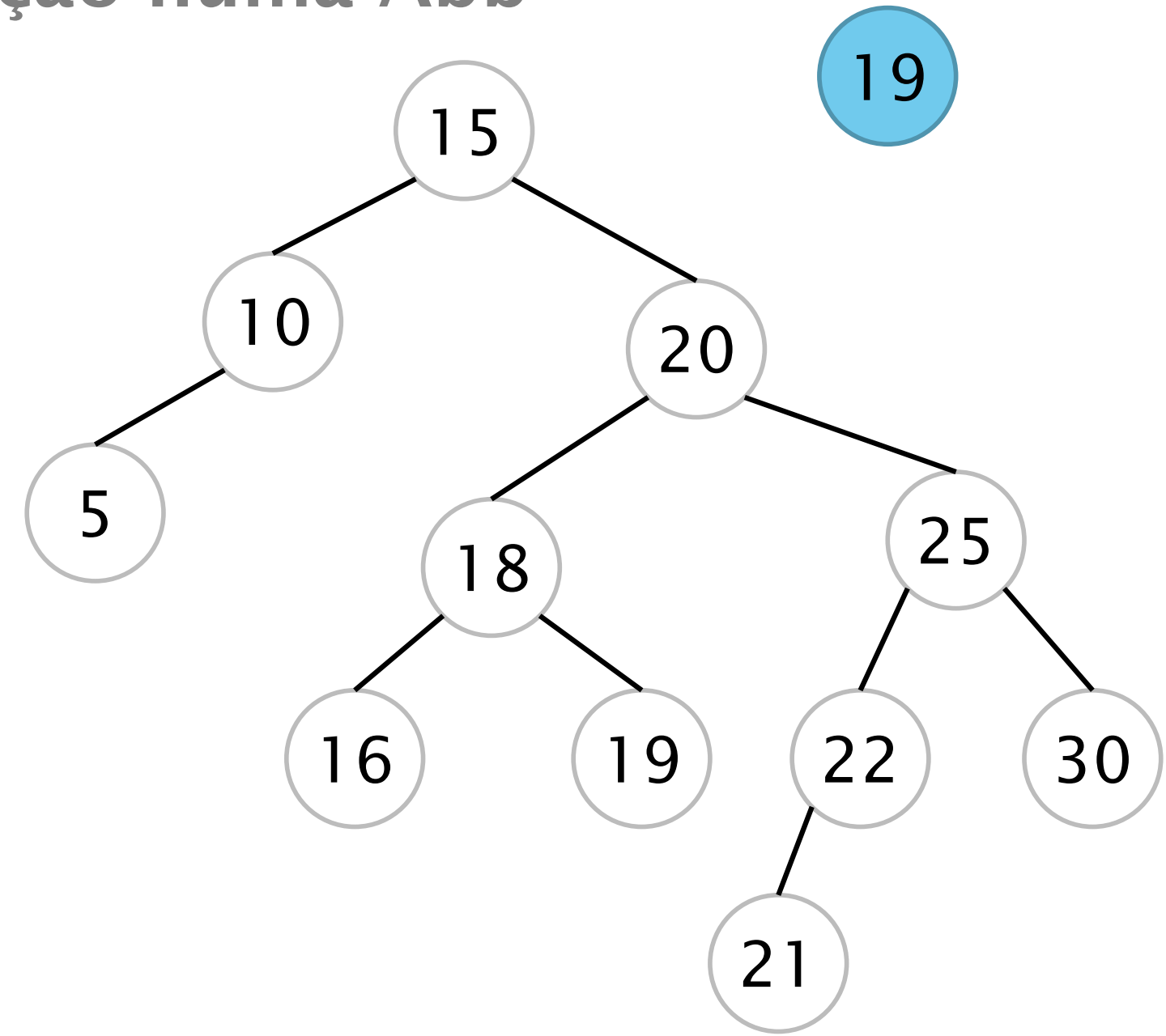
```
Node* Abb::insertrec(Node* a, int key) {
    if (a == nullptr) {
        a = new Node;
        a->_key = key;
        a->_left = a->_right = a->_up = nullptr;
    }
    if (key > a->_key) {
        a->_right = insertrec(a->_right, key);
        a->_right->_up = a;
        return a;
    } else if (key < a->_key) {
        a->_left = insertrec(a->_left, key);
        a->_left->_up = a;
    }
    return a;
}
```

Remoção de um nó de uma abb

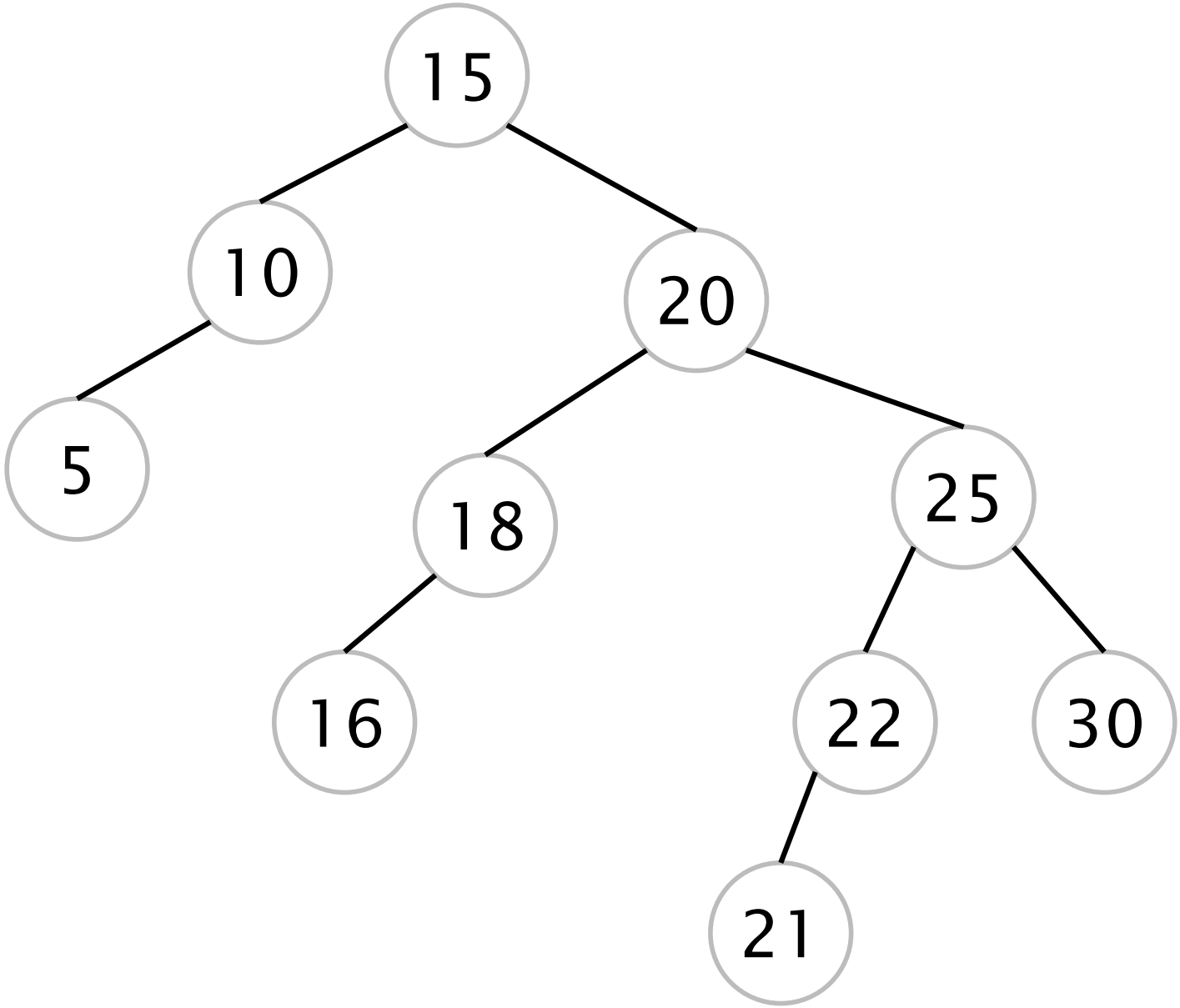
Três casos:

1. nó folha
2. nó possui uma sub-árvore
3. nó possui duas sub-árvores

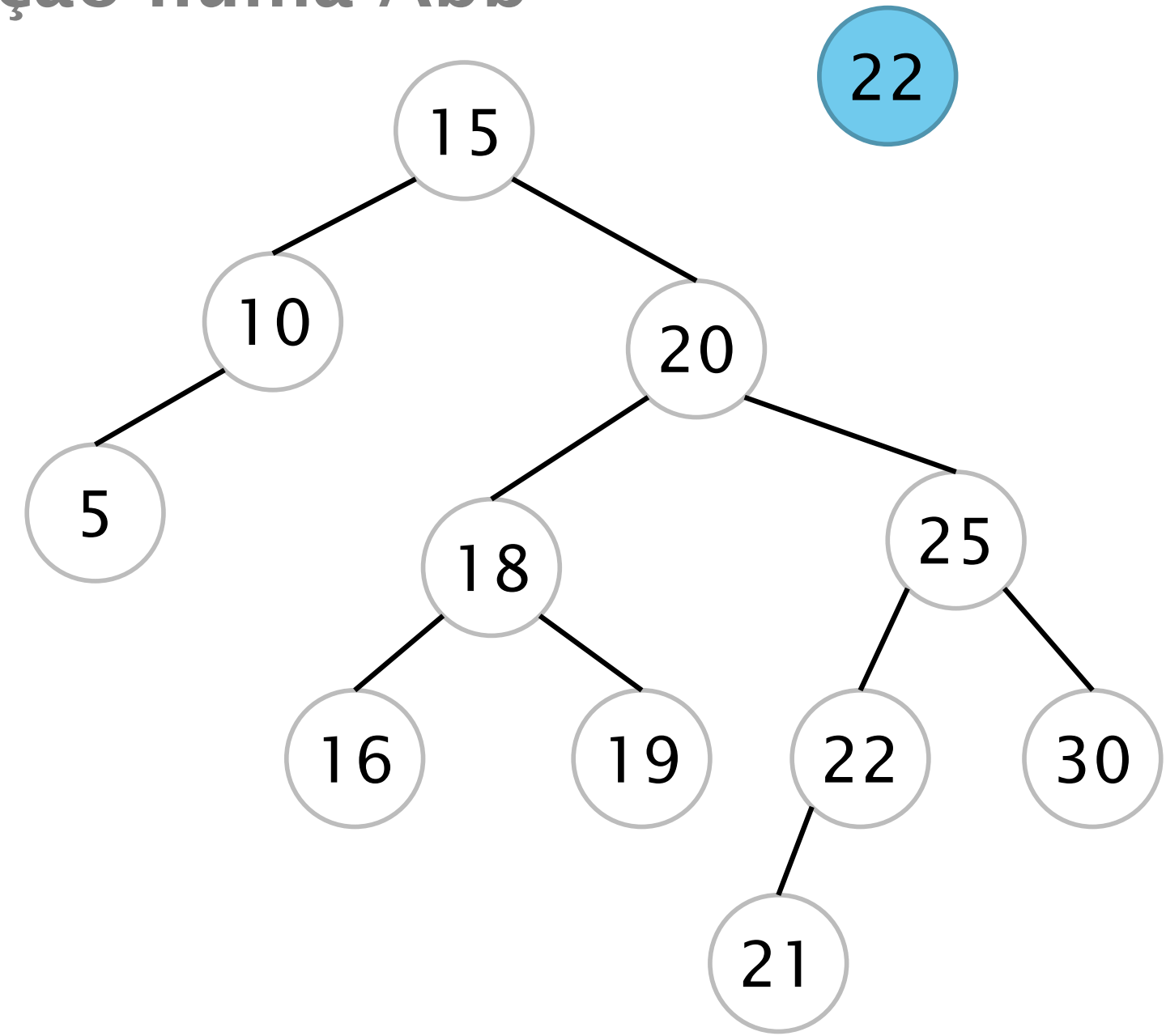
Remoção numa Abb



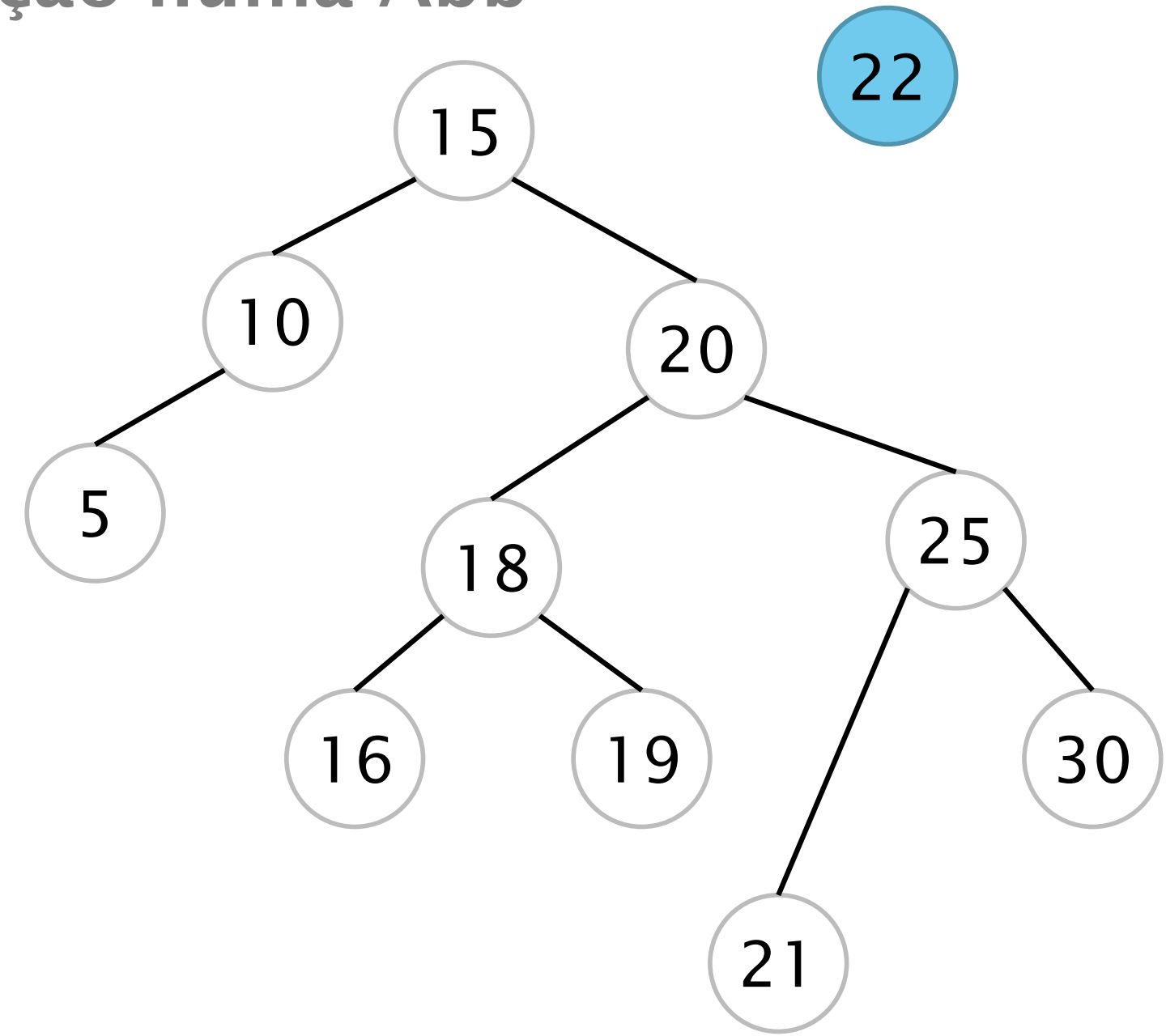
Remoção numa Abb



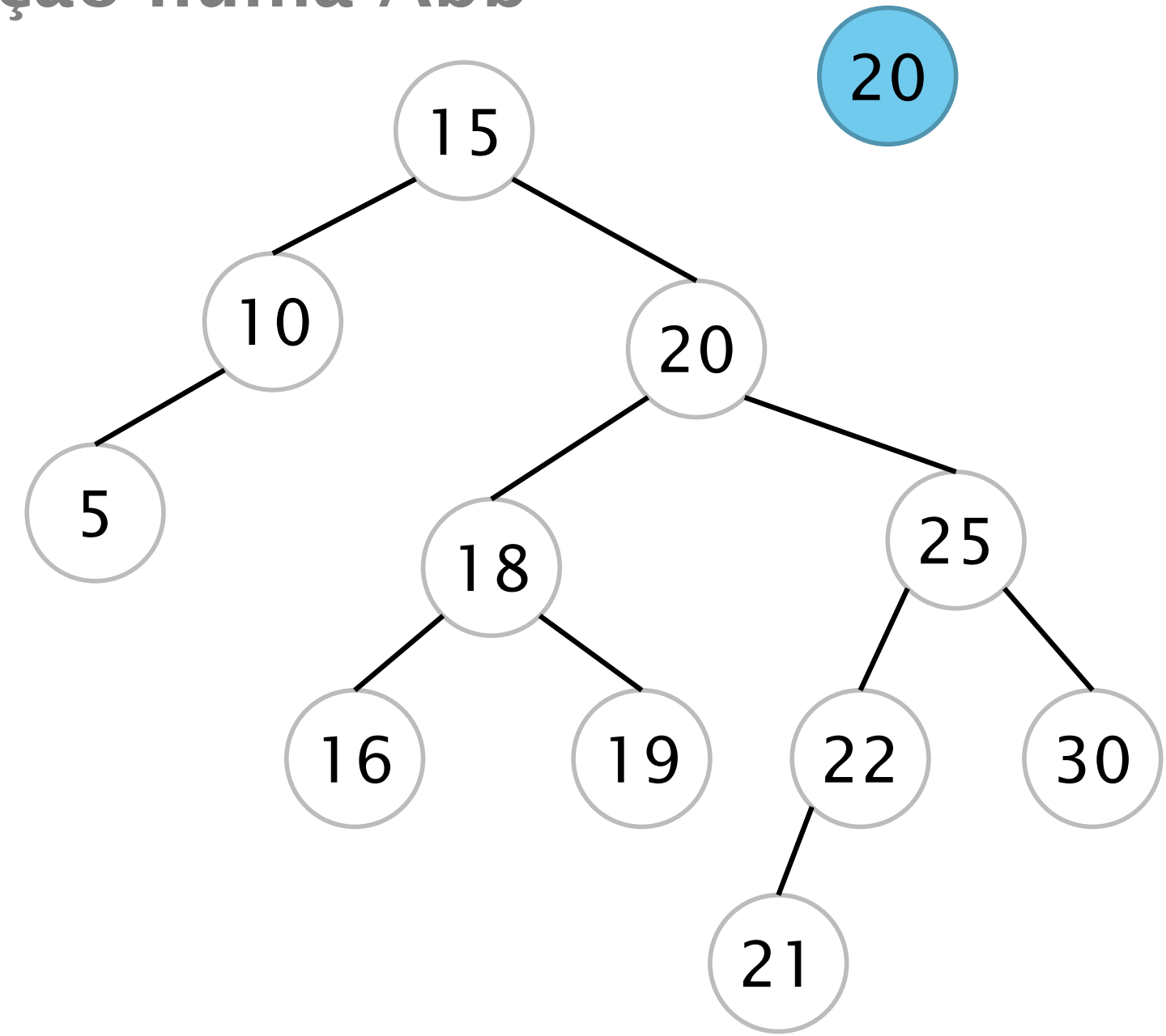
Remoção numa Abb



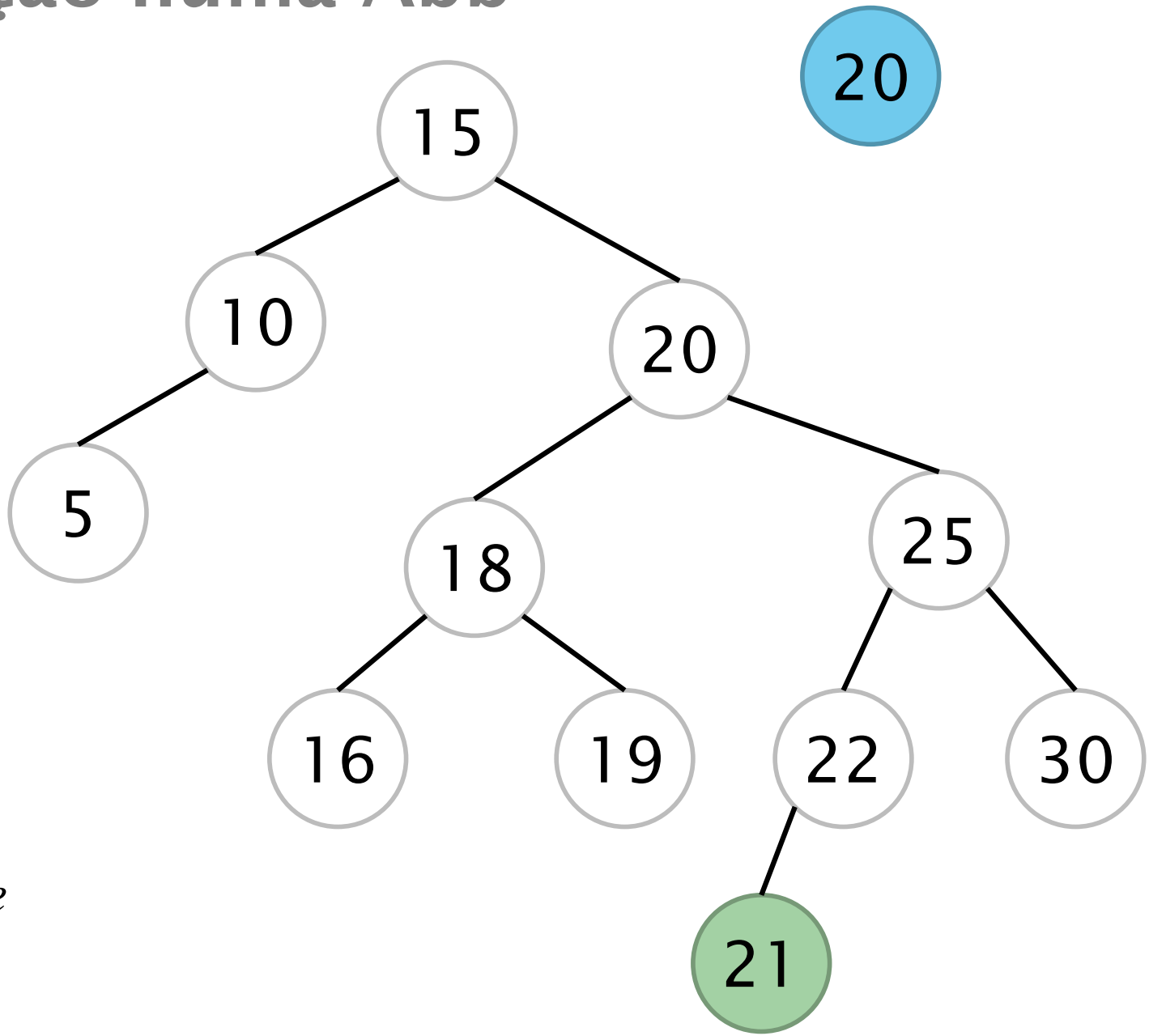
Remoção numa Abb



Remoção numa Abb

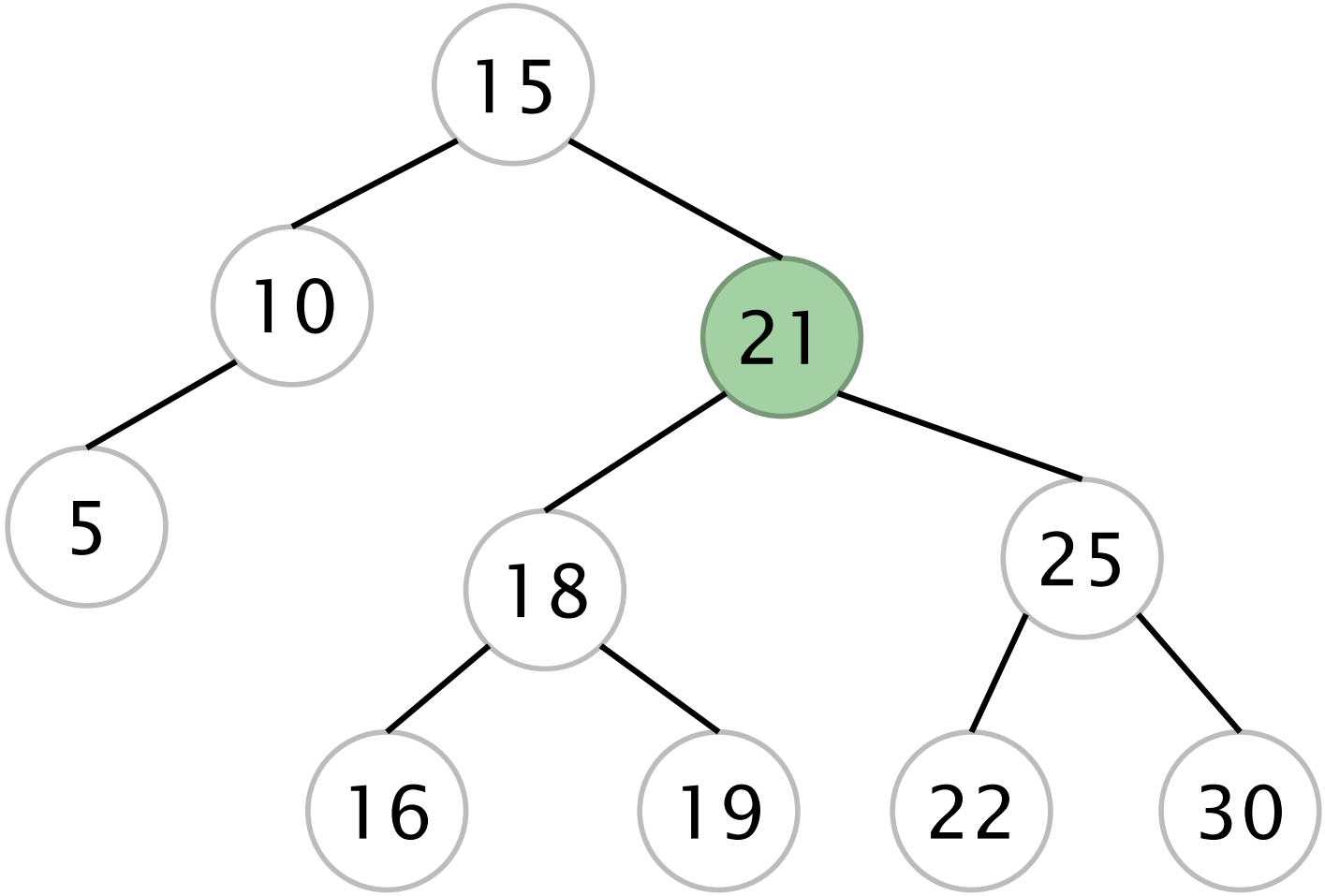


Remoção numa Abb

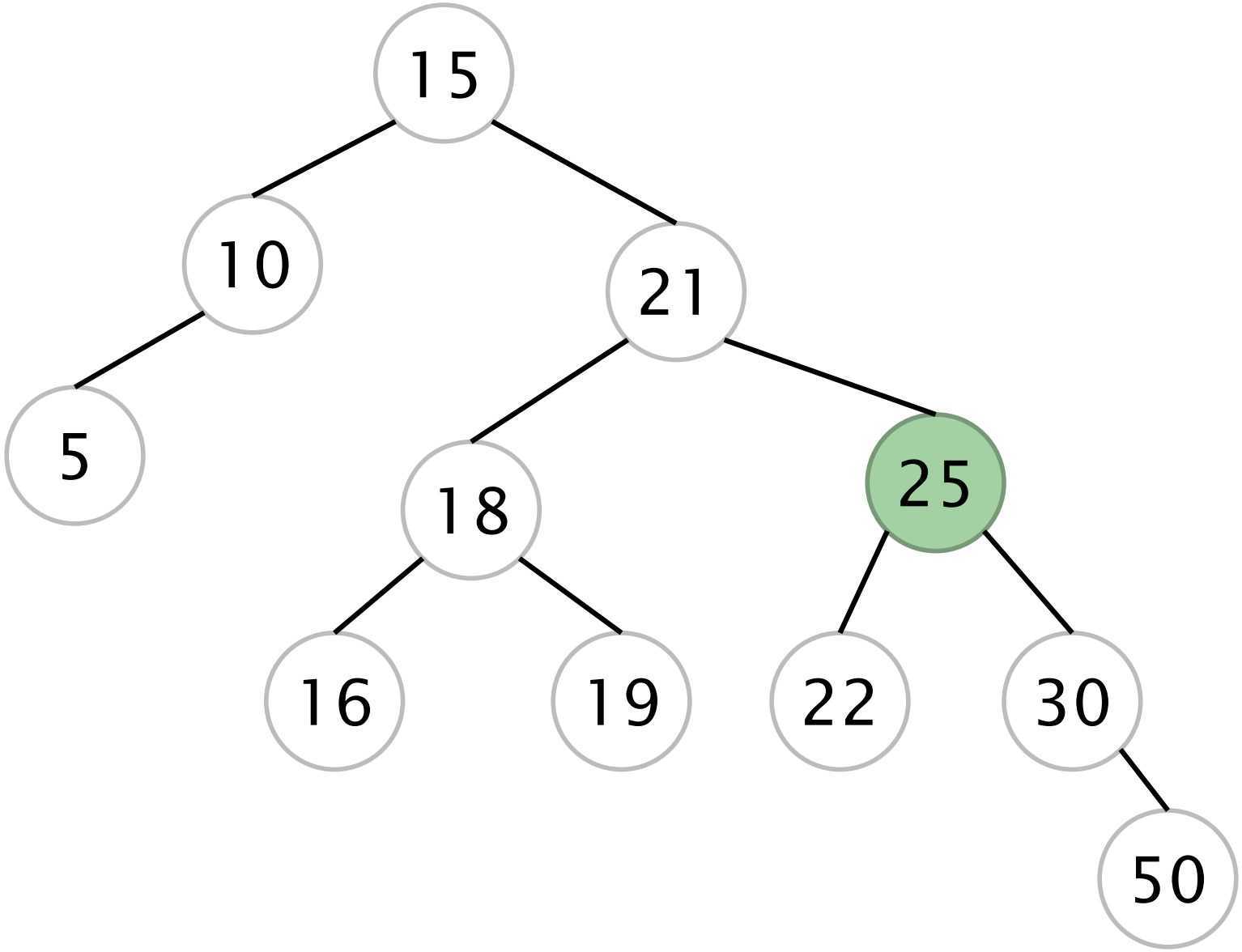


Quem é o sucessor de 20?

Remoção numa Abb



Remoção numa Abb



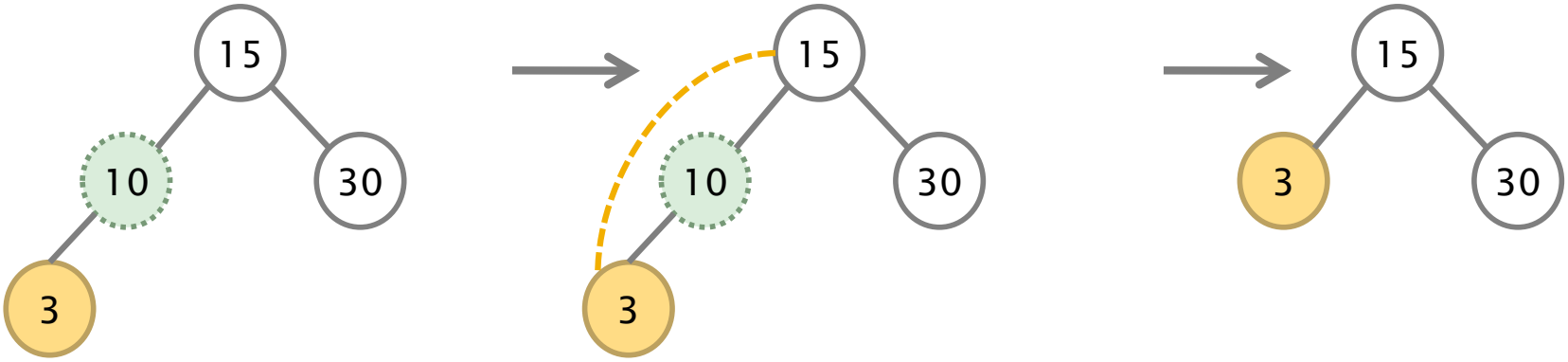
Remoção de um nó de uma abb

Três casos:

1. nó folha ou possui apenas uma sub-árvore

   promove a sub-árvore

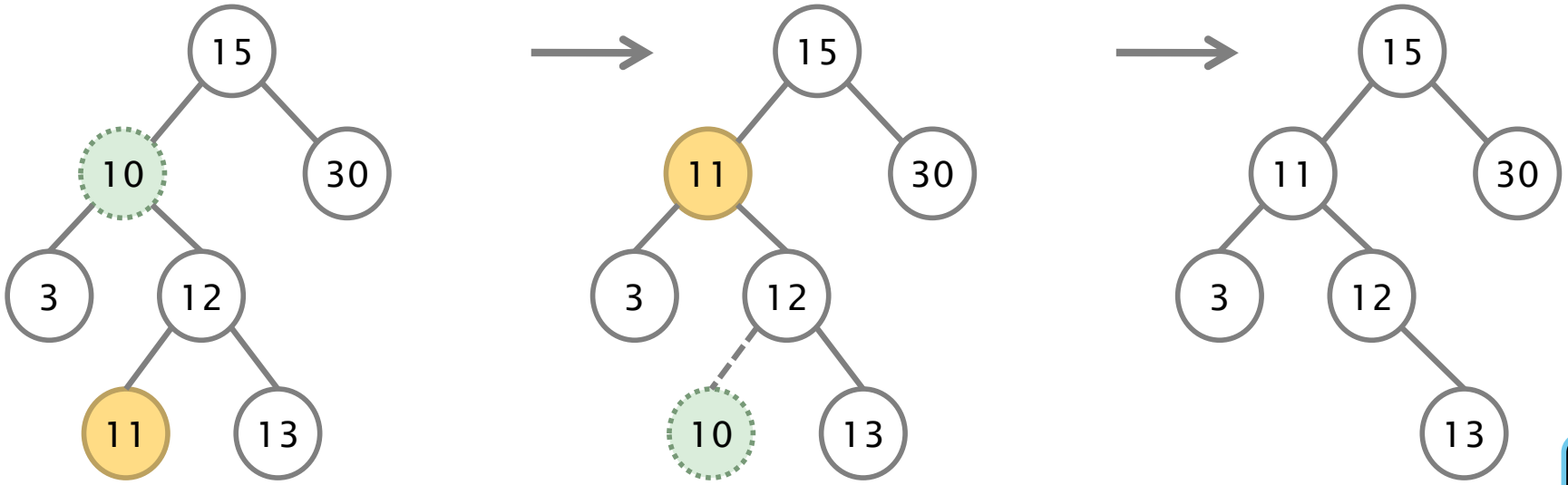
2. nó possui duas sub-árvores



Remoção de um nó de uma abb

Três casos:

1. nó folha
2. nó possui uma sub-árvore
3. nó possui duas sub-árvores
   1. coloque a informação do sucessor no nó
   2. remova o sucessor



© 2018 DI, PUC-Rio · Estruturas de Dados Avançadas – Versão C++

Remoção de um nó numa abb

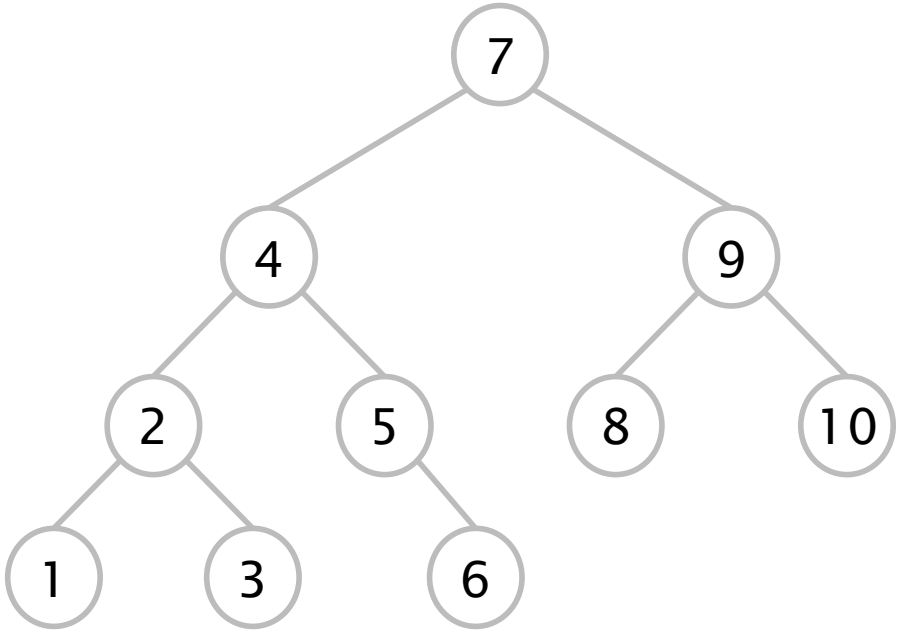
1. Ache o nó a ser removido
2. Se ele tiver um ou menos filhos, faça a ligação avô-neto
3. CC se ele tiver dois filhos, procure o sucessor, troque a info do nó pela do seu sucessor. Apague o sucessor.

Remoção de um nó numa abb (obs)

1. O sucessor é sempre o nó de menor chave da sub-arvore à direita
2. Ao retirar um nó temos que atualizar os campos de seu pai e de seu filho para que eles se referenciem

Exemplo

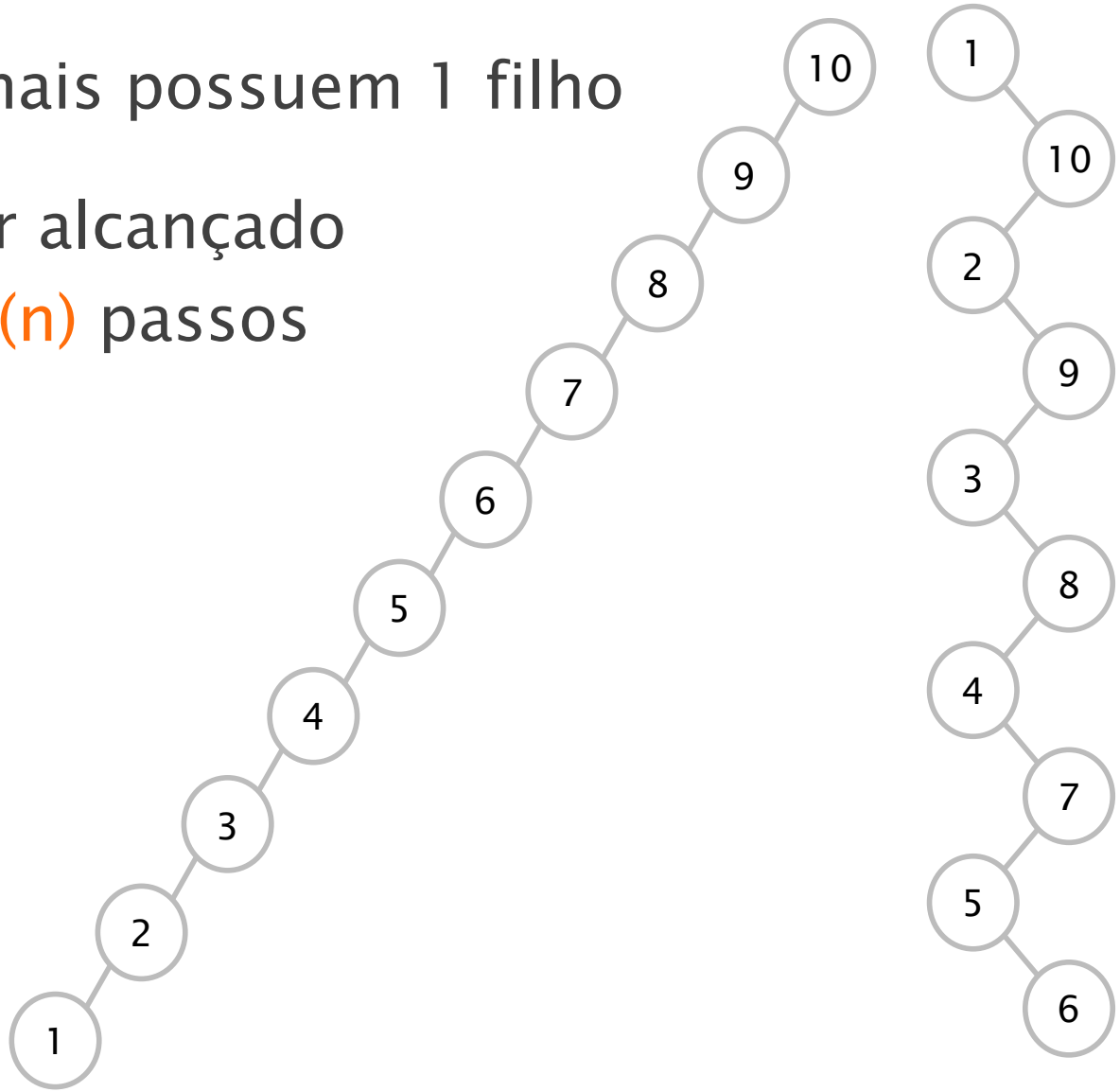
```
Abb a(7);
a.insert(4);
a.insert(9);
a.insert(2);
a.insert(5);
a.insert(8);
a.insert(10);
a.insert(1);
a.insert(3);
a.insert(6);
```



Árvore binária de busca degenerada

todos nós não terminais possuem 1 filho

qualquer nó pode ser alcançado
a partir da raiz em O(n) passos



Árvore binária de busca balanceada

$|he-hd| \leq 1$

he = altura da sub-árvore esquerda

hd = altura da sub-árvore direita

qualquer nó pode ser alcançado a partir da raiz em O(log(n)) passos

(quase) todos os nós não terminais têm dois filhos