EDA0001 – Estruturas de Dados

Árvores Binárias de Busca

Prof. Rui Jorge Tramontin Junior
Departamento de Ciência da Computação
UDESC / Joinville

Tópicos a serem apresentados

- Árvores;
  - Conceitos, aplicações, implementação;
- Árvores Binárias;
  - Conceitos, aplicações, implementação;
- **Árvores Binárias de Busca;**
  - **Aplicações e implementação;**
- Árvores AVL;
  - Conceitos e implementação.

Árvore Binária de Busca (ABB)

- **Definição**: árvore binária de busca (ABB) é uma *árvore binária* na qual, para cada nó *n*:

Árvore Binária de Busca (ABB)

- **Definição**: árvore binária de busca (ABB) é uma *árvore binária* na qual, para cada nó *n*:
  - A sua sub-árvore **à esquerda** contém somente nós com **valores menores** que o valor em *n*;

Árvore Binária de Busca (ABB)

- **Definição**: árvore binária de busca (ABB) é uma *árvore binária* na qual, para cada nó *n*:
  - A sua sub-árvore **à esquerda** contém somente nós com **valores menores** que o valor em *n*;
  - A sua sub-árvore **à direita** contém somente nós com **valores maiores** que o valor em *n*;

Árvore Binária de Busca (ABB)

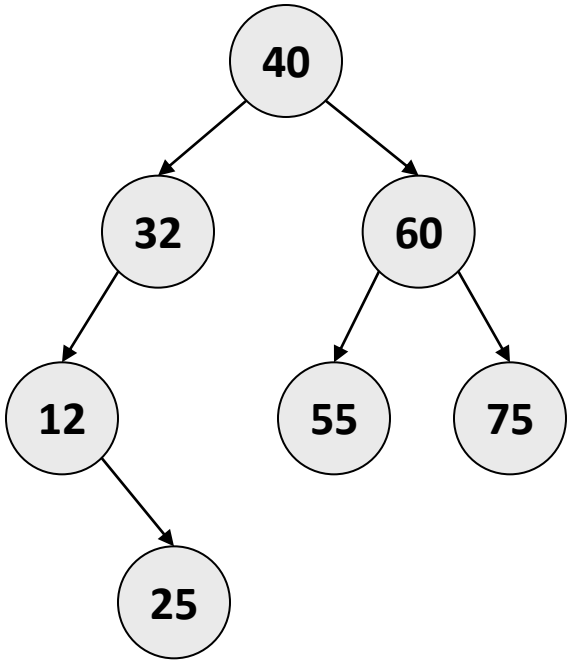
- **Definição**: árvore binária de busca (ABB) é uma *árvore binária* na qual, para cada nó *n*:
  - A sua sub-árvore **à esquerda** contém somente nós com **valores menores** que o valor em *n*;
  - A sua sub-árvore **à direita** contém somente nós com **valores maiores** que o valor em *n*;
  - Ambas as sub-árvores **também sejam ABB**.

Árvore Binária de Busca (ABB)

- **Definição**: árvore binária de busca (ABB) é uma *árvore binária* na qual, para cada nó *n*:
  - A sua sub-árvore **à esquerda** contém somente nós com **valores menores** que o valor em *n*;
  - A sua sub-árvore **à direita** contém somente nós com **valores maiores** que o valor em *n*;
  - Ambas as sub-árvores **também sejam ABB**.

- Os nós de uma ABB contêm **valores distintos**.

Exemplo de ABB



Aplicações de ABB

- **ABB** são usadas em problemas que necessitem de busca, onde dados são inseridos e removidos constantemente;

Aplicações de ABB

- **ABB** são usadas em problemas que necessitem de busca, onde dados são inseridos e removidos constantemente;

- Podem também ser usadas para implementar classes do tipo *Map* e *Set* em bibliotecas de linguagens de programação;

Implementação de ABB

- Mesmo modelo de nó de árvore binária:
  - Informação;
  - Ponteiros para os filhos à esquerda e à direita.

Implementação de ABB

- Mesmo modelo de nó de árvore binária:
  - Informação;
  - Ponteiros para os filhos à esquerda e à direita.

```
typedef struct noABB {
    int info;
    struct noABB *esq;
    struct noABB *dir;
}NoABB;
```

Operações Básicas

- Inserção;

Operações Básicas

- Inserção;
- Busca um valor;

Operações Básicas

- Inserção;
- Busca um valor;
- Encontrar o maior valor (ou o menor);

Operações Básicas

- Inserção;
- Busca um valor;
- Encontrar o maior valor (ou o menor);
- Remoção;

Operações Básicas

- Inserção;
- Busca um valor;
- Encontrar o maior valor (ou o menor);
- Remoção;
- Percurso (varredura nos nós).

INSERÇÃO

Inserção

- Deve obedecer as condições de uma ABP;

Inserção

- Deve obedecer as condições de uma ABP;
- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;

Inserção

- Deve obedecer as condições de uma ABP;
- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;
  - Se for maior, faz uma chamada recursiva para o **nó à direita**;

Inserção

- Deve obedecer as condições de uma ABP;

- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;
  - Se for maior, faz uma chamada recursiva para o **nó à direita**;
  - Se for menor, faz uma chamada recursiva para o **nó à esquerda**;

Inserção

- Deve obedecer as condições de uma ABP;

- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;
  - Se for maior, faz uma chamada recursiva para o **nó à direita**;
  - Se for menor, faz uma chamada recursiva para o **nó à esquerda**;
  - Se for igual, **nada acontece** (*valor já inserido*);

Inserção

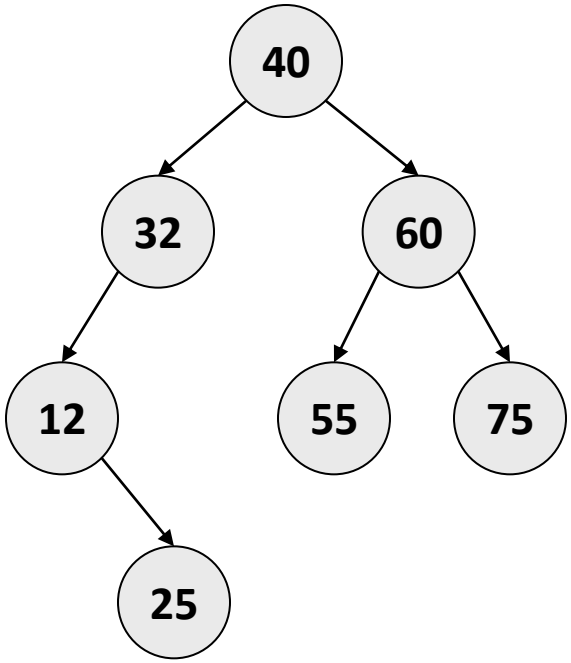
- Deve obedecer as condições de uma ABP;

- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;
  - Se for maior, faz uma chamada recursiva para o **nó à direita**;
  - Se for menor, faz uma chamada recursiva para o **nó à esquerda**;
  - Se for igual, **nada acontece** (*valor já inserido*);
  - Quando chegar num nó nulo (*NULL*), significa que chegou na posição **onde o valor será inserido** (ou árvore está vazia);

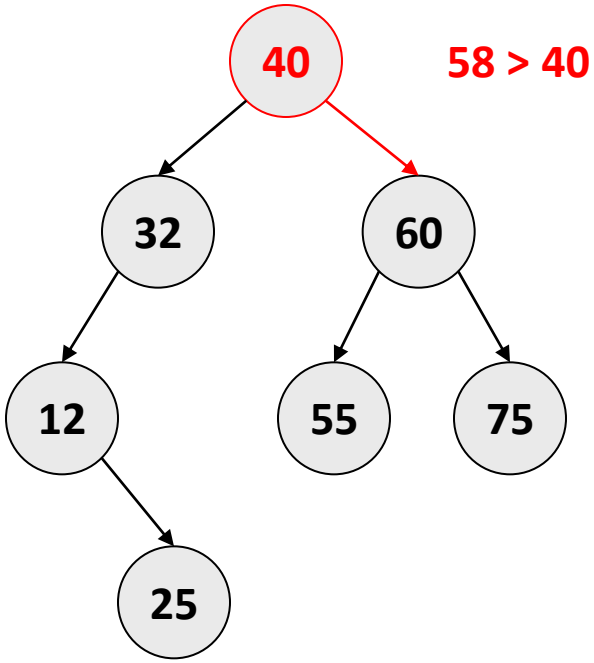
Inserção

- Deve obedecer as condições de uma ABP;
- A partir do nó raiz, a função compara o valor a ser inserido com o valor no nó;
  - Se for maior, faz uma chamada recursiva para o **nó à direita**;
  - Se for menor, faz uma chamada recursiva para o **nó à esquerda**;
  - Se for igual, **nada acontece** (*valor já inserido*);
  - Quando chegar num nó nulo (*NULL*), significa que chegou na posição **onde o valor será inserido** (ou árvore está vazia);
- O novo nó sempre será um nó folha.

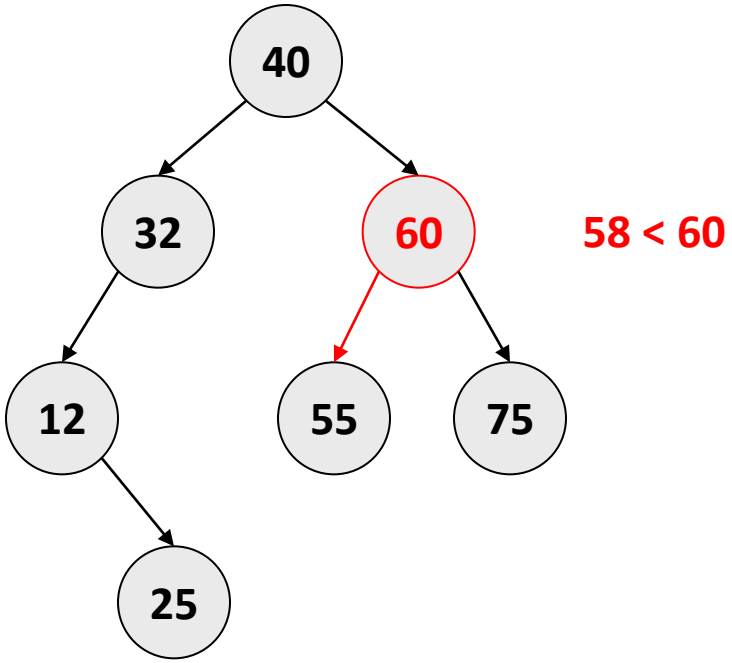
Exemplo: inserção do valor 58



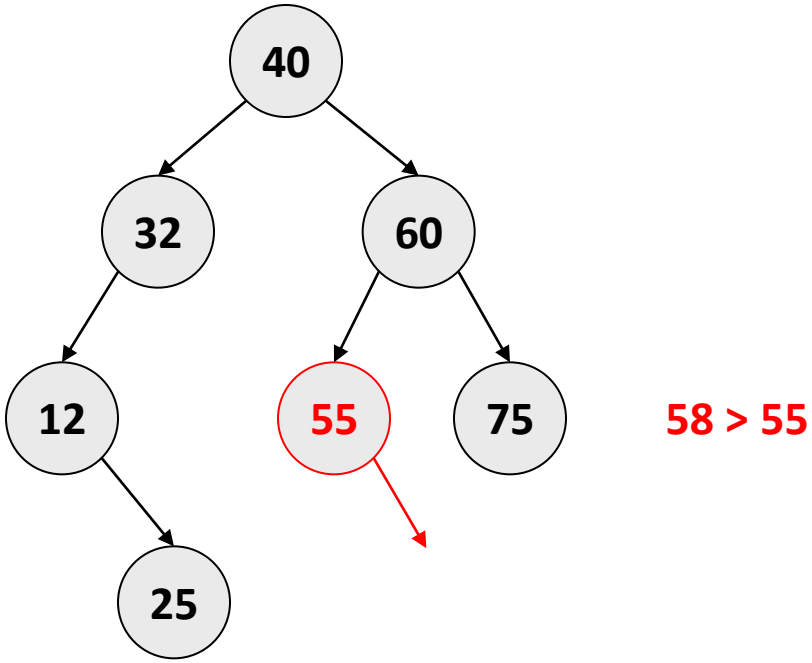
Exemplo: inserção do valor 58



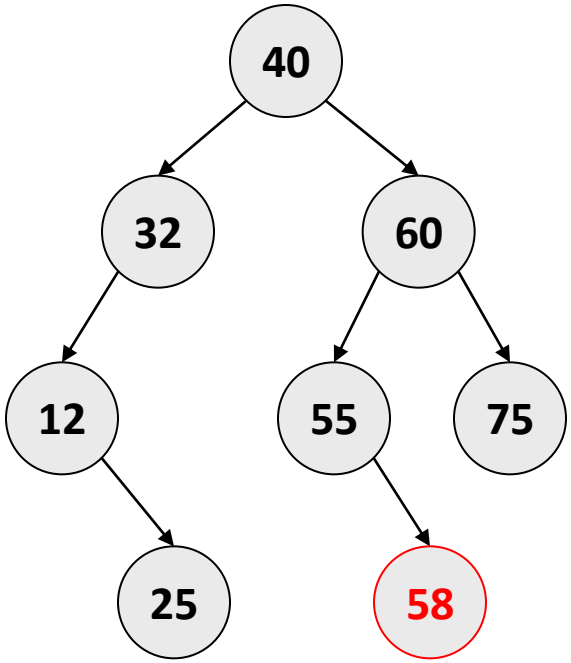
Exemplo: inserção do valor 58



Exemplo: inserção do valor 58



Exemplo: inserção do valor 58



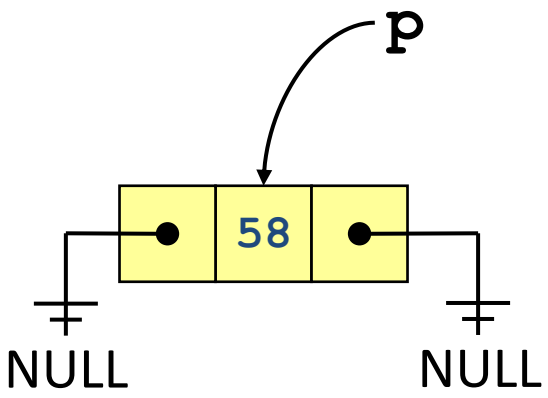
Insere à direta do 55

Inserção: função

```
NoABB *insere_abb( NoABB *n, int info ){
  if( n == NULL ) // Árvore vazia (fim percurso)
    return aloca_no( info );

  if( info < n->info ) // Perccore à esquerda.
    n->esq = insere_abb( n->esq, info );
  else
    if( info > n->info ) // Perccore à direita.
      n->dir = insere_abb( n->dir, info );

  return n; // Retorna o próprio nó.
}
```

Função para alocação do nó

```
NoABB *aloca_no( int info ){
   NoABB *p = malloc( sizeof(NoABB) );
   if( p == NULL)
      return NULL;

   p->info = info;
   p->esq = NULL;
   p->dir = NULL;
   return p;
}
```



BUSCA

Busca

- É semelhante ao algortimo da inserção;

Busca

- É semelhante ao algortimo da inserção;
- Percorre recursivamente, comparando a chave de busca com os valores nos nós da árvores;

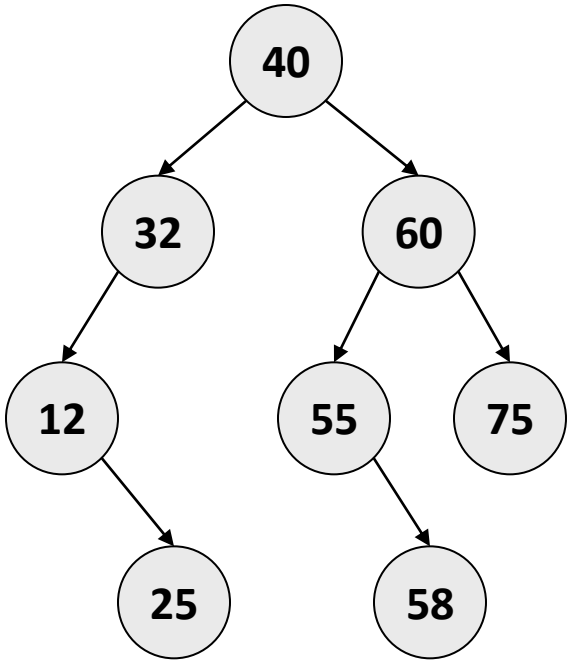
Busca

- É semelhante ao algortimo da inserção;
- Percorre recursivamente, comparando a chave de busca com os valores nos nós da árvores;
- Ao final, retorna o nó onde a chave se encontra, ou *NULL* caso não encontre;

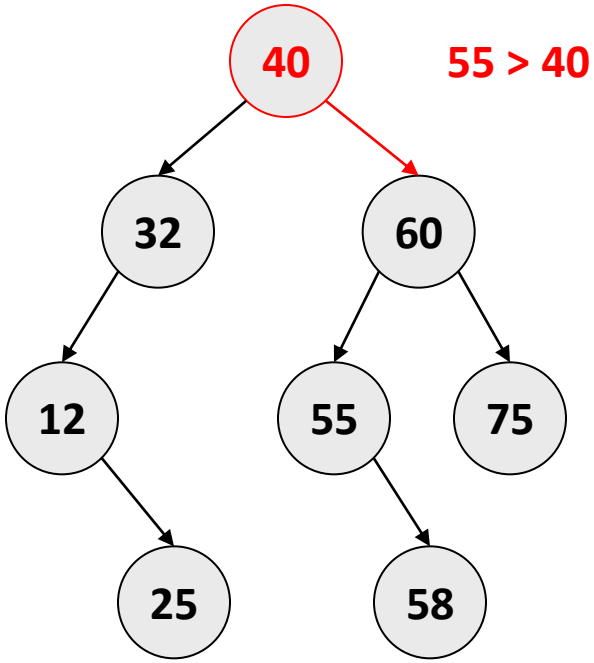
Busca

- É semelhante ao algortimo da inserção;
- Percorre recursivamente, comparando a chave de busca com os valores nos nós da árvores;
- Ao final, retorna o nó onde a chave se encontra, ou *NULL* caso não encontre;
- Algoritmo equivalente a uma **busca binária** em um vetor ordenado.

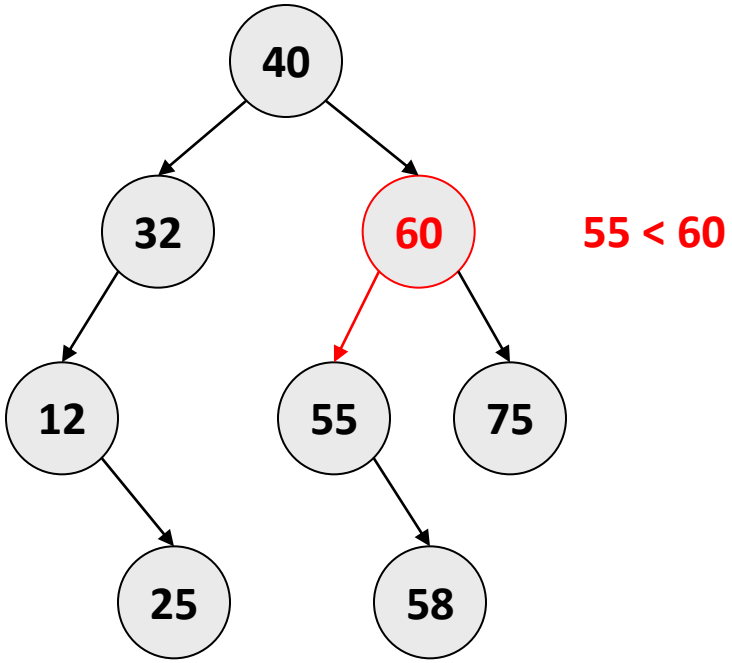
Exemplo: busca do valor 55



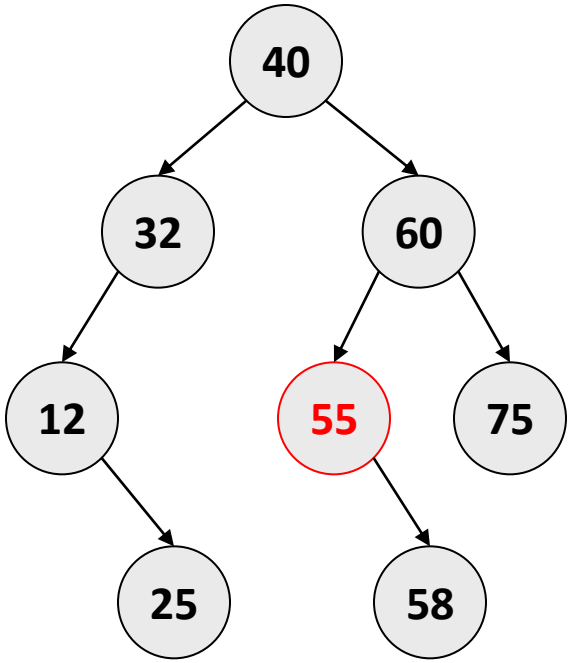
Exemplo: busca do valor 55



Exemplo: busca do valor 55



Exemplo: busca do valor 55



Encontrado!

Busca: função

```
NoABB *busca_abb( NoABB *n, int chave ){
  if( n == NULL )
    return NULL;

  if( chave < n->info )
    return busca_abb( n->esq, chave );

  if( chave > n->info )
    return busca_abb( n->dir, chave );

  return n;
}
```

ENCONTRAR O MAIOR VALOR

Encontrar o maior valor

- Como os nós estão em ordem, basta percorrer pela **subárvore à direita** de cada nó a partir da raiz;

Encontrar o maior valor

- Como os nós estão em ordem, basta percorrer pela **subárvore à direita** de cada nó a partir da raiz;

- O processo recursivo termina quando encontra-se um nó **sem filho à direita**;

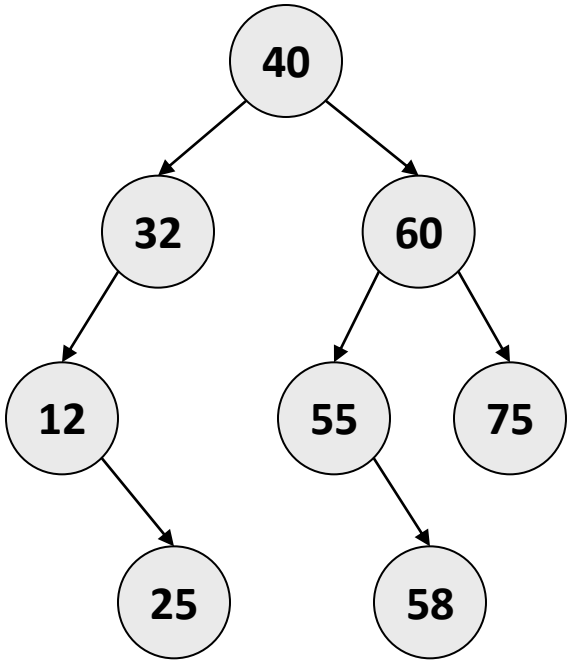
Encontrar o maior valor

- Como os nós estão em ordem, basta percorrer pela **subárvore à direita** de cada nó a partir da raiz;

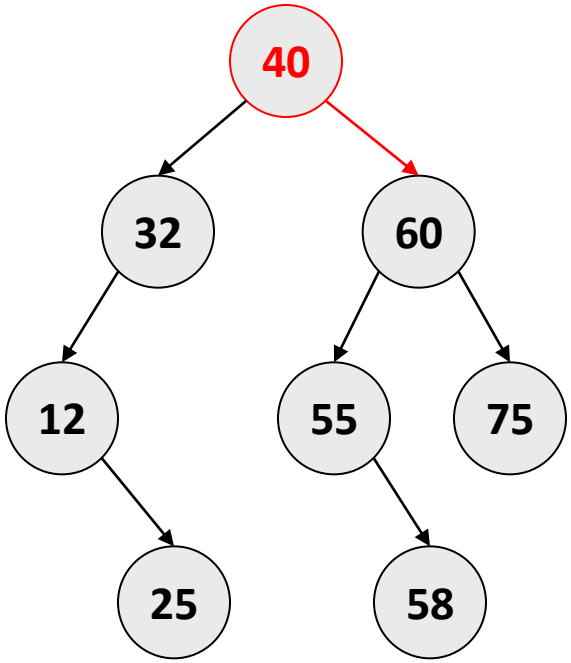
- O processo recursivo termina quando encontra-se um nó **sem filho à direita**;

- O mesmo pode ser feito para encontrar o **menor**, mas o percurso é pela **esquerda**.

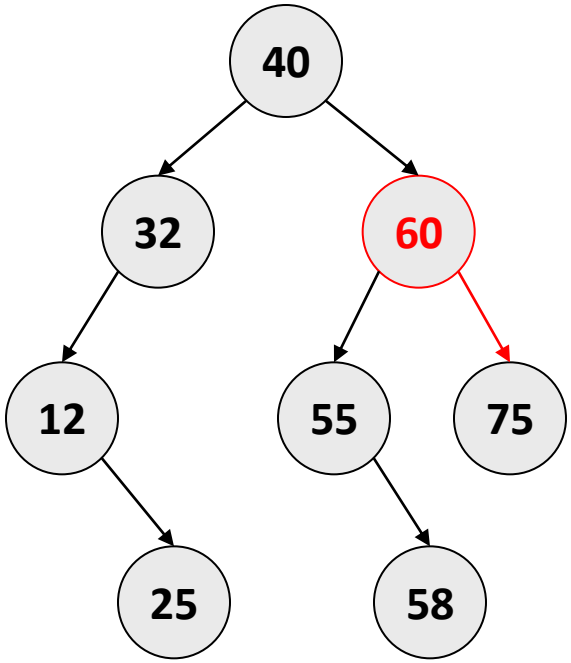
Exemplo: maior valor



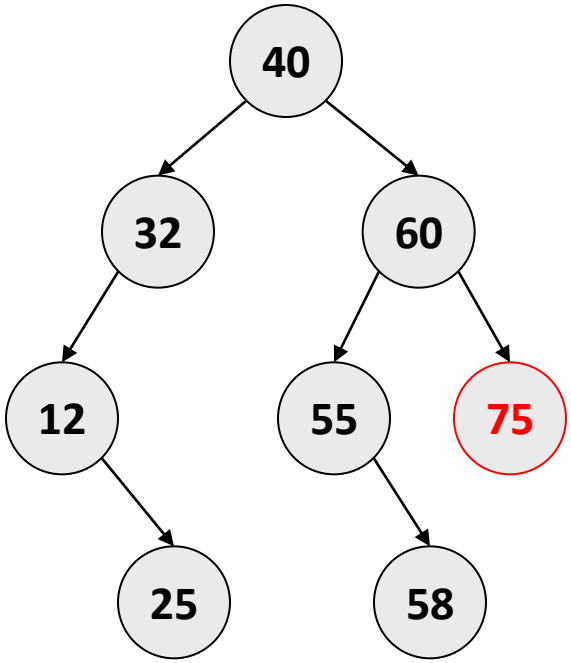
Exemplo: maior valor



Exemplo: maior valor



Exemplo: maior valor



Maior valor!

Encontrar o maior valor: função

```
NoABB *maior( NoABB *n ){
  if( n == NULL)
    return NULL;

  if( n->dir == NULL) // Achou o maior!
    return n;
  else
    return maior( n->dir ); // Continua...
}
```

REMOÇÃO

Remoção

- A remoção de um nó deve considerar um dos seguintes casos:

Remoção

- A remoção de um nó deve considerar um dos seguintes casos:
  - Nó folha (nenhum filho);

Remoção

- A remoção de um nó deve considerar um dos seguintes casos:
  - Nó folha (nenhum filho);
  - Nó com somente um filho;

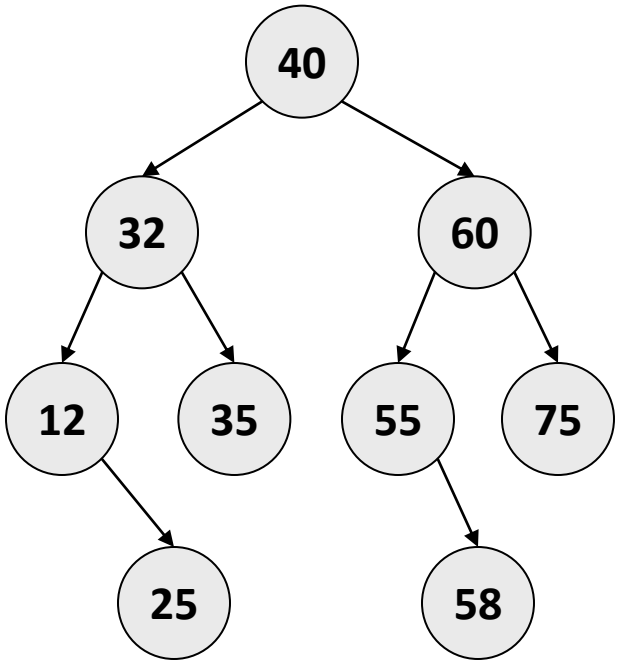
Remoção

- A remoção de um nó deve considerar um dos seguintes casos:
  - Nó folha (nenhum filho);
  - Nó com somente um filho;
  - Nó com dois filhos;

REMOÇÃO DE UM NÓ FOLHA

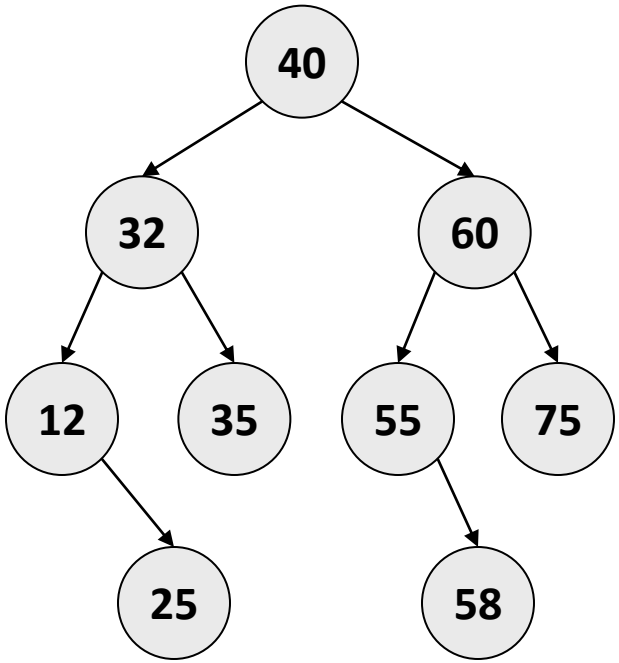
Remoção de um nó folha

- Basta encontrá-lo e removê-lo (trivial);



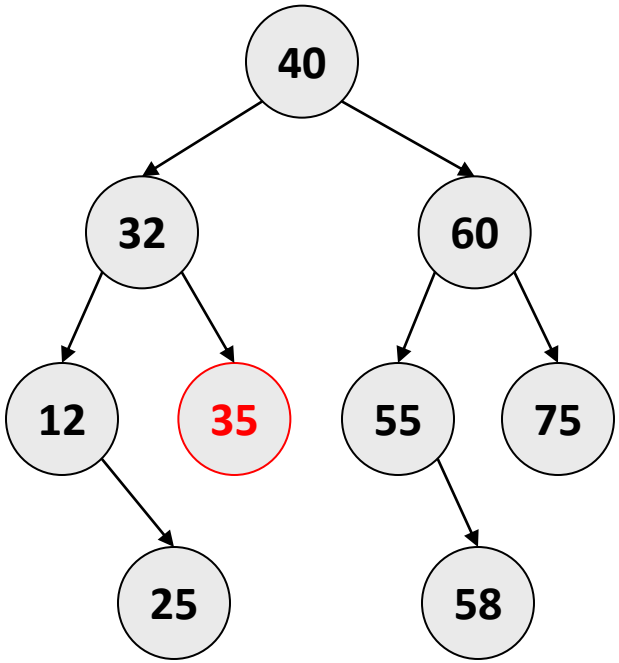
Remoção de um nó folha

- Basta encontrá-lo e removê-lo (trivial);
- **Exemplo**: remove 35.



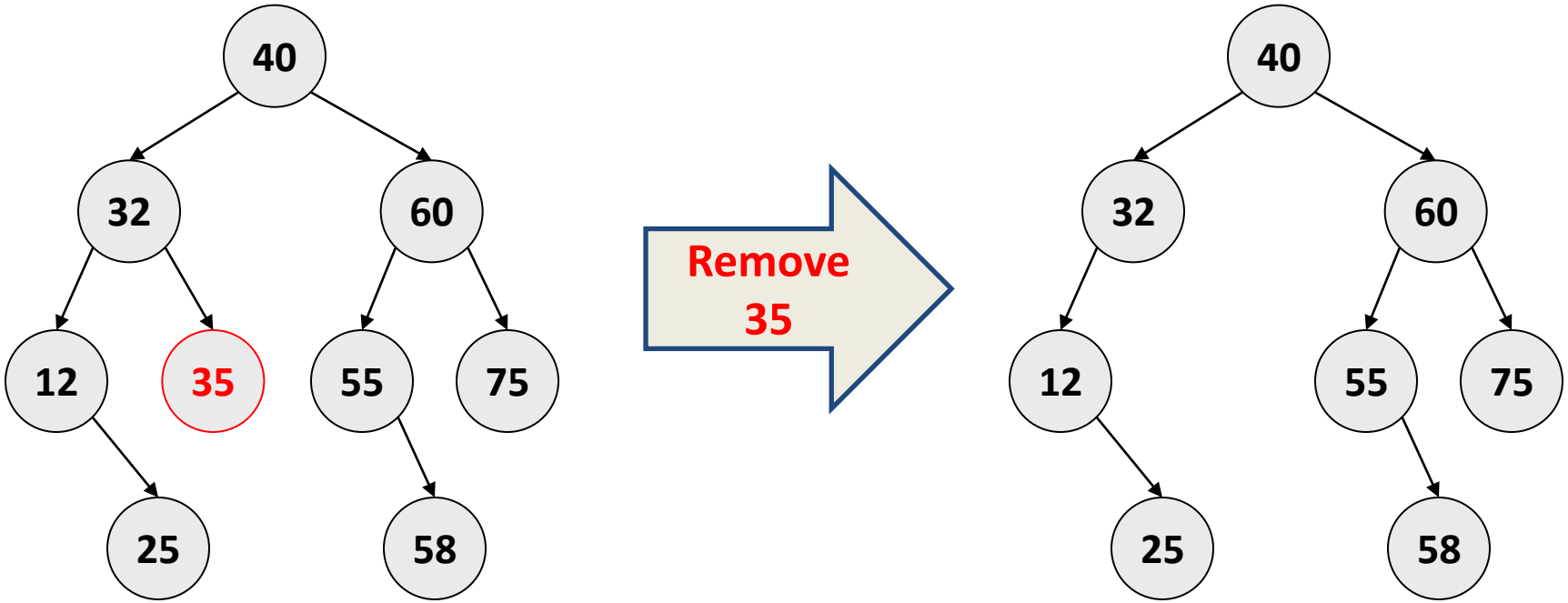
Remoção de um nó folha

- Basta encontrá-lo e removê-lo (trivial);
- **Exemplo**: remove 35.



Remoção de um nó folha

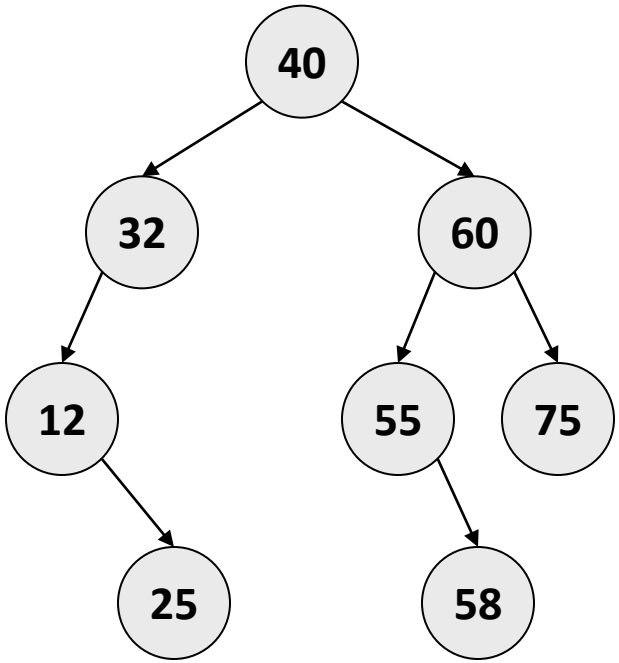
- Basta encontrá-lo e removê-lo (trivial);
- **Exemplo**: remove 35.



REMOÇÃO DE UM NÓ COM UM FILHO

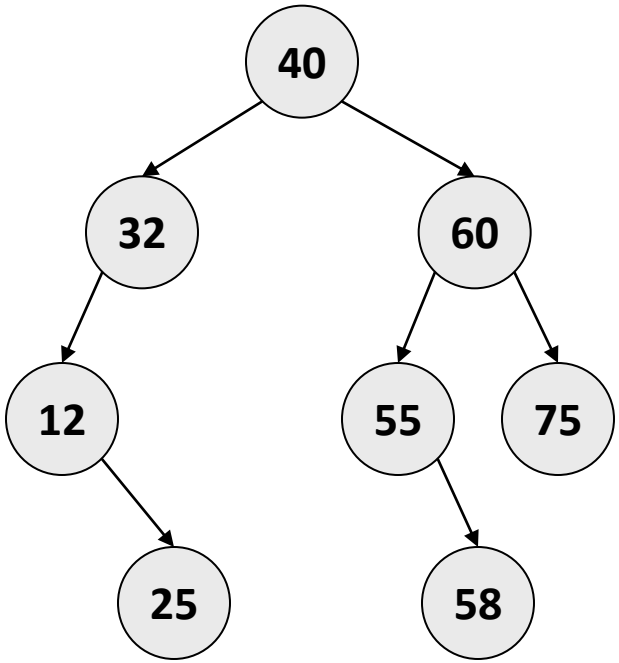
Remoção de um nó com um filho

- Basta removê-lo; seu filho (subárvore) ocupa seu lugar;



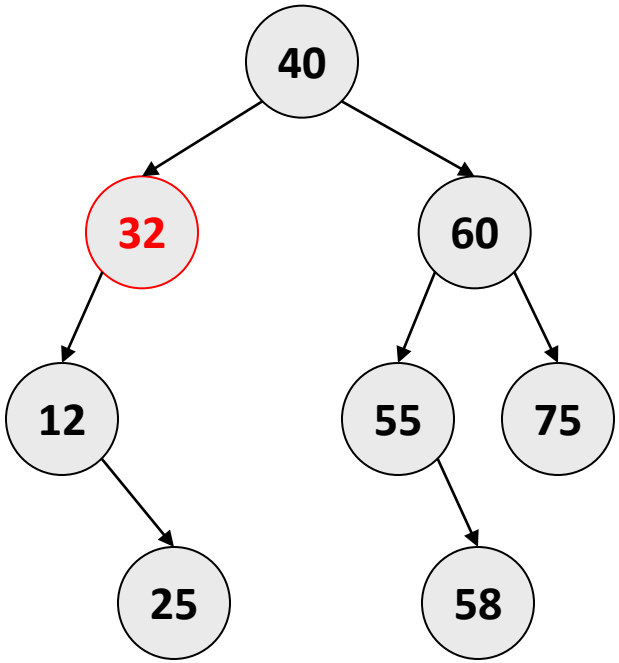
Remoção de um nó com um filho

- Basta removê-lo; seu filho (subárvore) ocupa seu lugar;
- **Exemplo**: remove 32.



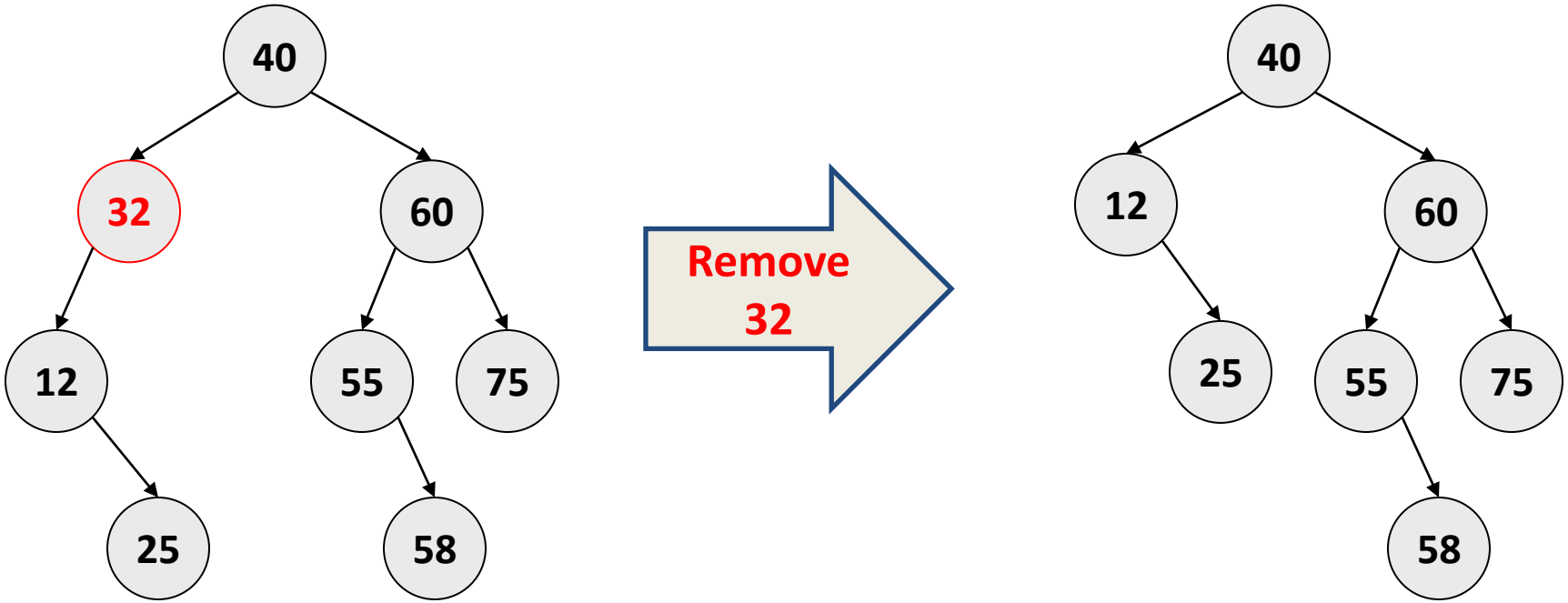
Remoção de um nó com um filho

- Basta removê-lo; seu filho (subárvore) ocupa seu lugar;
- **Exemplo**: remove 32.



Remoção de um nó com um filho

- Basta removê-lo; seu filho (subárvore) ocupa seu lugar;
- **Exemplo**: remove 32.



REMOÇÃO DE UM NÓ COM DOIS FILHOS

Remoção de um nó com dois filhos

- Os casos de remoção de nó folha ou com um filho são triviais;

Remoção de um nó com dois filhos

- Os casos de remoção de nó folha ou com um filho são triviais;

- A remoção de um nó ***n*** com dois filhos envolve os seguintes passos:

Remoção de um nó com dois filhos

- Os casos de remoção de nó folha ou com um filho são triviais;

- A remoção de um nó ***n*** com dois filhos envolve os seguintes passos:
  - Encontre o **maior valor à esquerda** de ***n*** (ou o **menor valor à direita** de ***n***);

Remoção de um nó com dois filhos

- Os casos de remoção de nó folha ou com um filho são triviais;

- A remoção de um nó ***n*** com dois filhos envolve os seguintes passos:
  - Encontre o **maior valor à esquerda** de ***n*** (ou o **menor valor à direita** de ***n***);
  - **Copie** o valor encontrado para o nó ***n***;

Remoção de um nó com dois filhos

- Os casos de remoção de nó folha ou com um filho são triviais;

- A remoção de um nó ***n*** com dois filhos envolve os seguintes passos:
  - Encontre o **maior valor à esquerda** de ***n*** (ou o **menor valor à direita** de ***n***);
  - **Copie** o valor encontrado para o nó ***n***;
  - **Remova** o nó selecionado.

Remoção de um nó com dois filhos

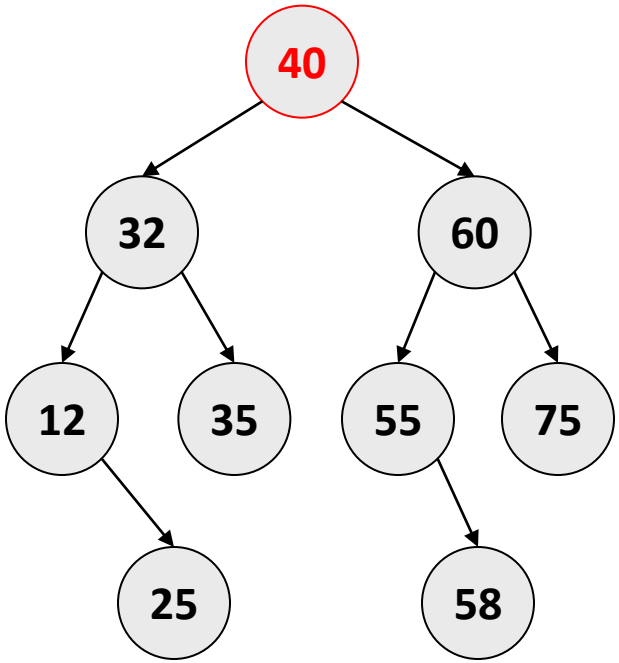
- Os casos de remoção de nó folha ou com um filho são triviais;

- A remoção de um nó ***n*** com dois filhos envolve os seguintes passos:
  - Encontre o **maior valor à esquerda** de ***n*** (ou o **menor valor à direita** de ***n***);
  - **Copie** o valor encontrado para o nó ***n***;
  - **Remova** o nó selecionado.

- O valor removido será sempre um nó folha ou nó com um filho.

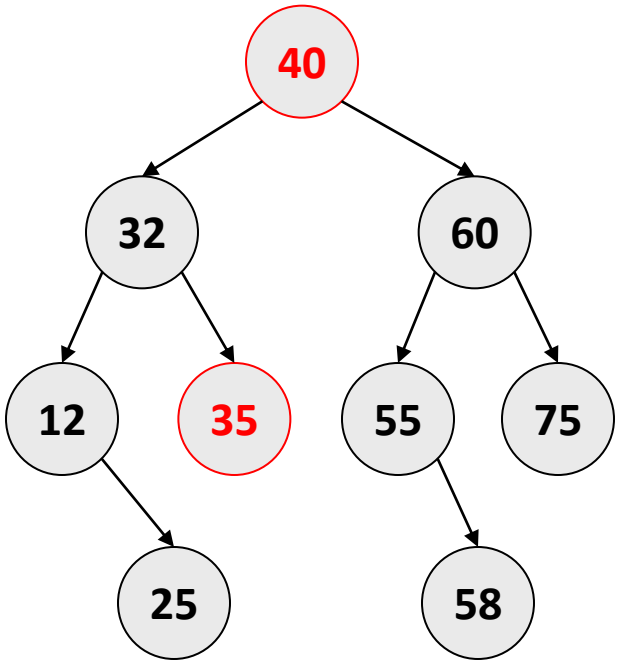
Remoção de um nó com dois filhos

- **Exemplo 1**: remove 40 (maior da esquerda).



Remoção de um nó com dois filhos

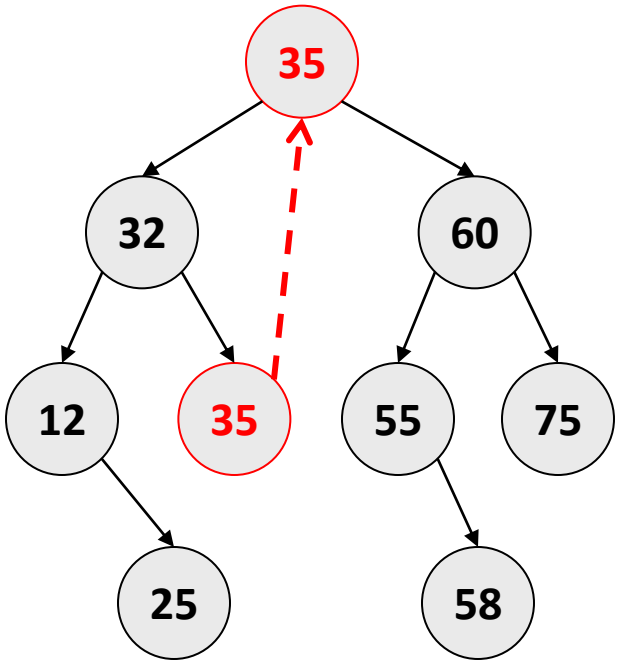
- **Exemplo 1**: remove 40 (maior da esquerda).
- **Passo 1**: encontrar maior valor à esquerda.



Maior valor à esquerda do 40

Remoção de um nó com dois filhos

- **Exemplo 1**: remove 40 (maior da esquerda).
- **Passo 2**: copiar valor encontrado.

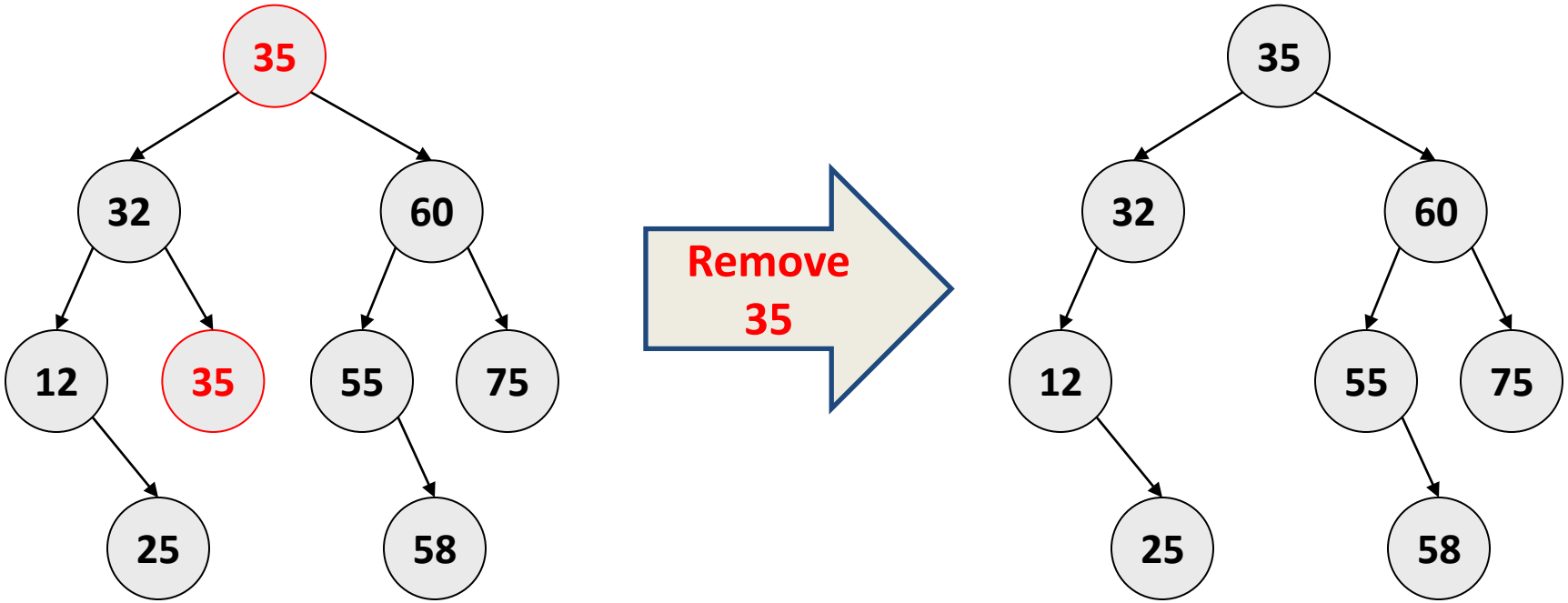


Copia o valor

Remoção de um nó com dois filhos

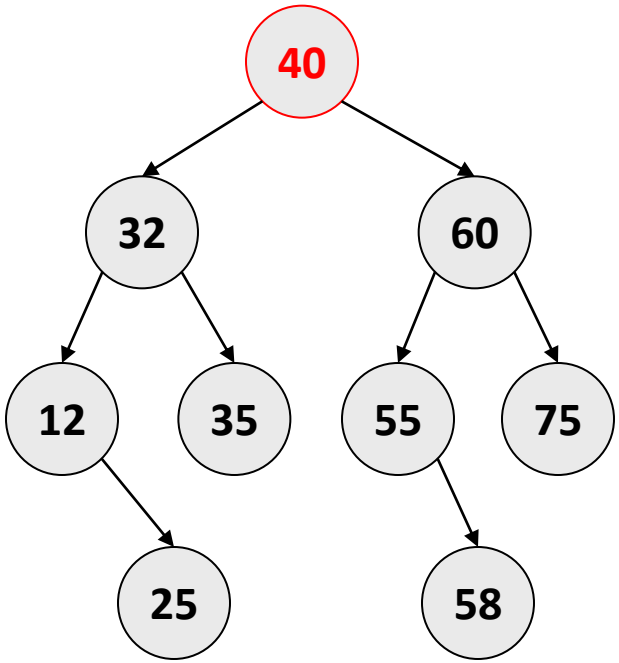
- **Exemplo 1**: remove 40 (maior da esquerda).

- **Passo 3**: remover valor encontrado.



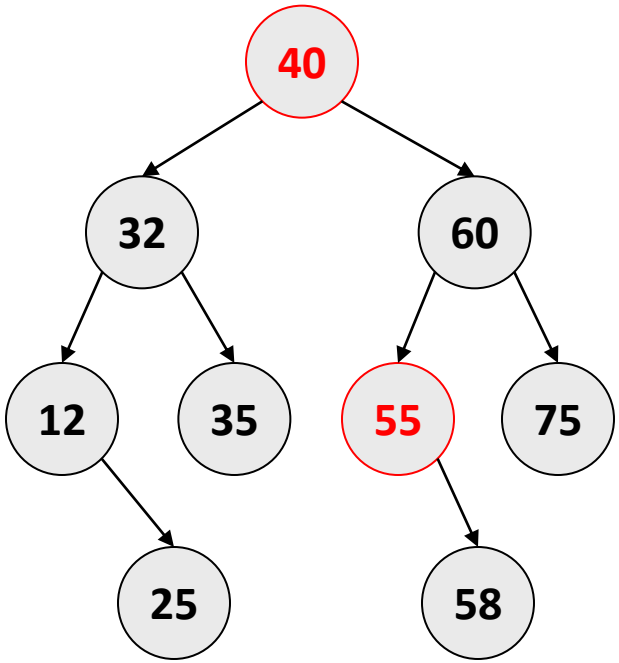
Remoção de um nó com dois filhos

- **Exemplo 1**: remove 40 (menor da direita).



Remoção de um nó com dois filhos

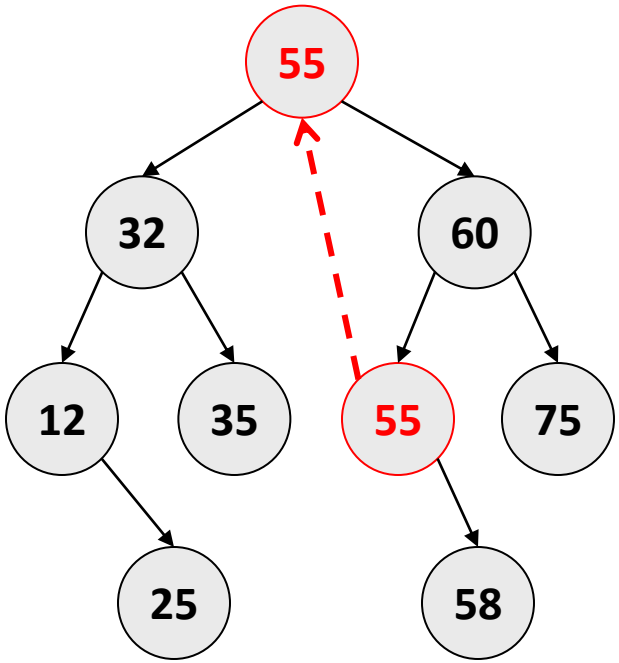
- **Exemplo 1**: remove 40 (menor da direita).
- **Passo 1**: encontrar menor valor à direita.



Menor valor à direita do 40

Remoção de um nó com dois filhos

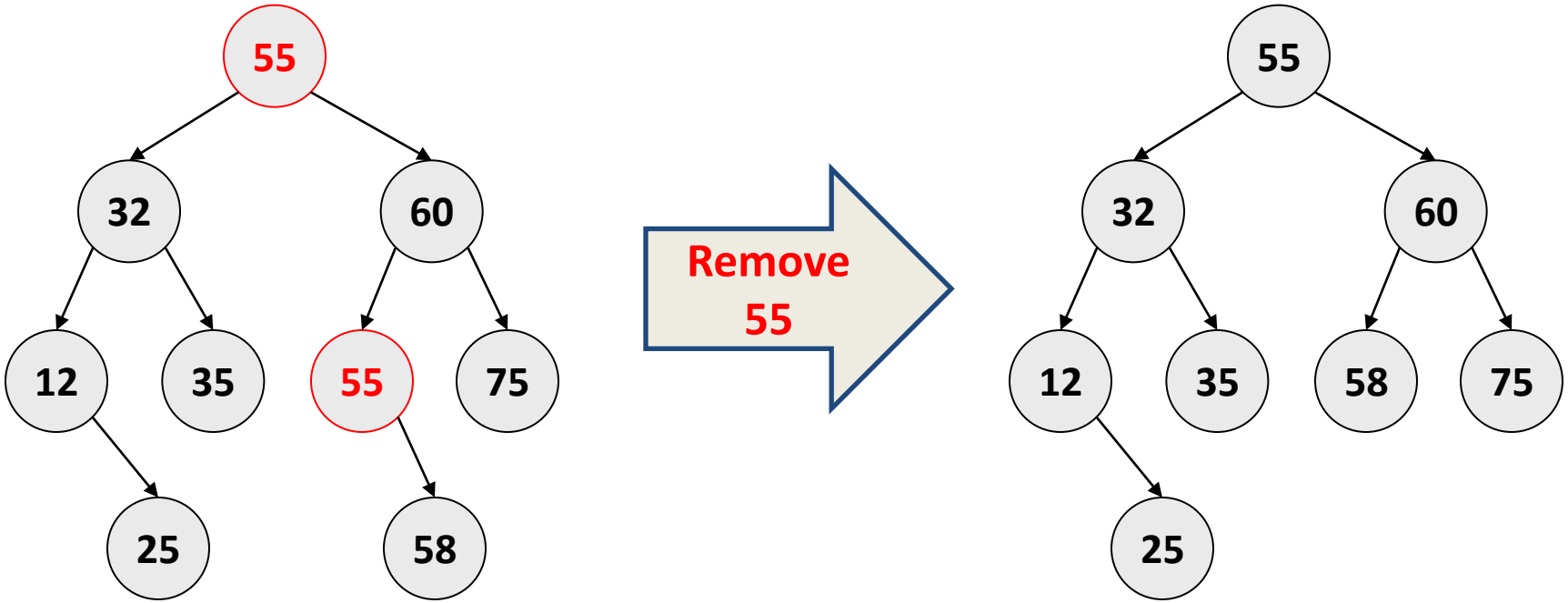
- **Exemplo 1**: remove 40 (menor da direita).
- **Passo 2**: copiar valor encontrado.



Copia o valor

Remoção de um nó com dois filhos

- **Exemplo 1**: remove 40 (menor da direita).
- **Passo 3**: remover valor encontrado.



Remoção: função 1 (busca o nó)

```
NoABB *remove_abb( NoABB *n, int info ){
  if( n == NULL)
    return n;

  if( info == n->info )
    return remove_no( n );

  if( info < n->info )
    n->esq = remove_abb( n->esq, info );
  else
    n->dir = remove_abb( n->dir, info );

  return n;
}
```

Remoção: função 2 (remoção do nó)

```
NoABB *remove_no( NoABB *n ){
  NoABB *aux;
  if( n->esq == NULL ){ // 1 filho à dir. ou nó folha
    aux = n->dir;
    free( n );
    return aux;
  }
  if( n->dir == NULL ){ // 1 filho à esq.
    aux = n->esq;
    free( n );
    return aux;
  }
  // Nó com 2 filhos.
  NoABB *m = maior( n->esq ); // Maior à esquerda.
  n->info = m->info;
  n->esq = remove_abb( n->esq, m->info );
  return n;
}
```

PERCURSO

Percurso

- Uma operação muito comum em árvores em geral é o **percurso**, que significa passar por todos os nós, pelo menos uma vez;

Percurso

- Uma operação muito comum em árvores em geral é o **percurso**, que significa passar por todos os nós, pelo menos uma vez;

- O conceito de *visitar* significa executar uma operação com a informação armazenada no nó, por exemplo, **imprimir** seu conteúdo;

Percurso

- Uma operação muito comum em árvores em geral é o **percurso**, que significa passar por todos os nós, pelo menos uma vez;

- O conceito de *visitar* significa executar uma operação com a informação armazenada no nó, por exemplo, **imprimir** seu conteúdo;

- Na operação de percorrer a árvore pode-se passar por alguns nós mais de uma vez, porém sem visitá-los.

Percurso

- Uma árvore é uma estrutura não sequêncial, diferentemente de uma lista, por exemplo;

Percurso

- Uma árvore é uma estrutura não sequêncial, diferentemente de uma lista, por exemplo;

- Não existe ordem natural para percorrer árvores e, portanto, pode-se escolher diferentes maneiras para tal;

Percurso

- Uma árvore é uma estrutura não sequêncial, diferentemente de uma lista, por exemplo;

- Não existe ordem natural para percorrer árvores e, portanto, pode-se escolher diferentes maneiras para tal;

- Há duas formas gerais de percurso em árvores:

Percurso

- Uma árvore é uma estrutura não sequêncial, diferentemente de uma lista, por exemplo;

- Não existe ordem natural para percorrer árvores e, portanto, pode-se escolher diferentes maneiras para tal;

- Há duas formas gerais de percurso em árvores:
  - **Em profundidade**: pré-ordem, em-ordem e pós-ordem;

Percurso

- Uma árvore é uma estrutura não sequêncial, diferentemente de uma lista, por exemplo;

- Não existe ordem natural para percorrer árvores e, portanto, pode-se escolher diferentes maneiras para tal;

- Há duas formas gerais de percurso em árvores:
  - **Em profundidade**: pré-ordem, em-ordem e pós-ordem;
  - **Em largura**: percorre-se cada nível de cada vez.

Explicação dos Percursos

- Explicação detalhadas dos percursos e exemplos estão em uma apresentação em anexo;

- O conteúdo foi separado para facilitar o estudo posterior.

CONSIDERAÇÕES

Considerações

- Implementações pequenas (recursivas);

Considerações

- Implementações pequenas (recursivas);
- É possível implementar de maneira iterativa (com o uso de *pilhas*);

Considerações

- Implementações pequenas (recursivas);
- É possível implementar de maneira iterativa (com o uso de *pilhas*);
- Quando a ABB está **balanceada**, os algoritmos são eficientes → *O(log n)*;

Considerações

- Implementações pequenas (recursivas);
- É possível implementar de maneira iterativa (com o uso de *pilhas*);
- Quando a ABB está **balanceada**, os algoritmos são eficientes → *O(log n)*;
- Porém, uma ABB não consegue garantir o balanceamento!

Problemas de balanceamento

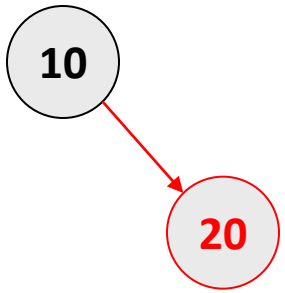
- Exemplo: inserção dos valores 10, 20, 35, 40, 55;

Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;

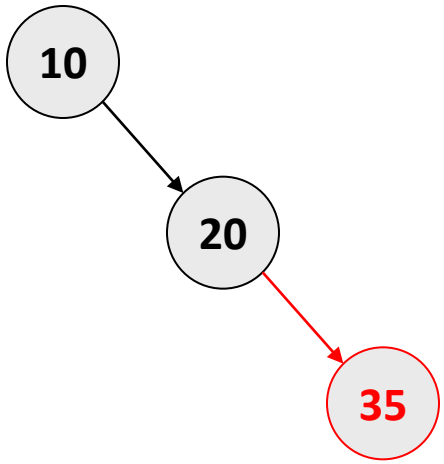
Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;



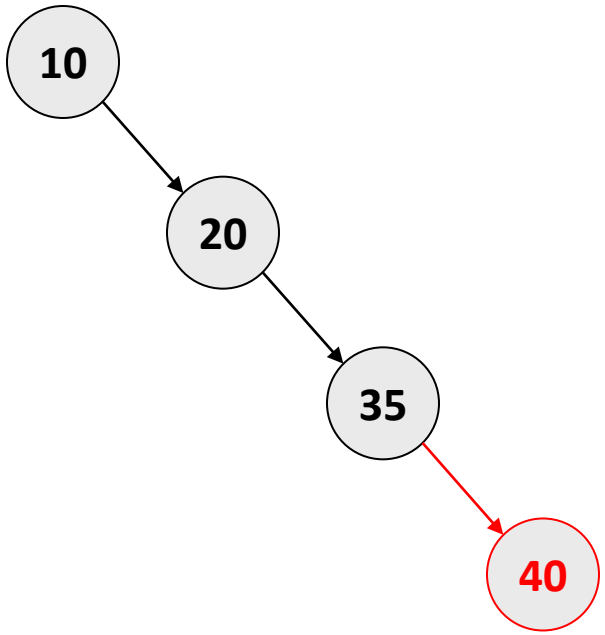
Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;



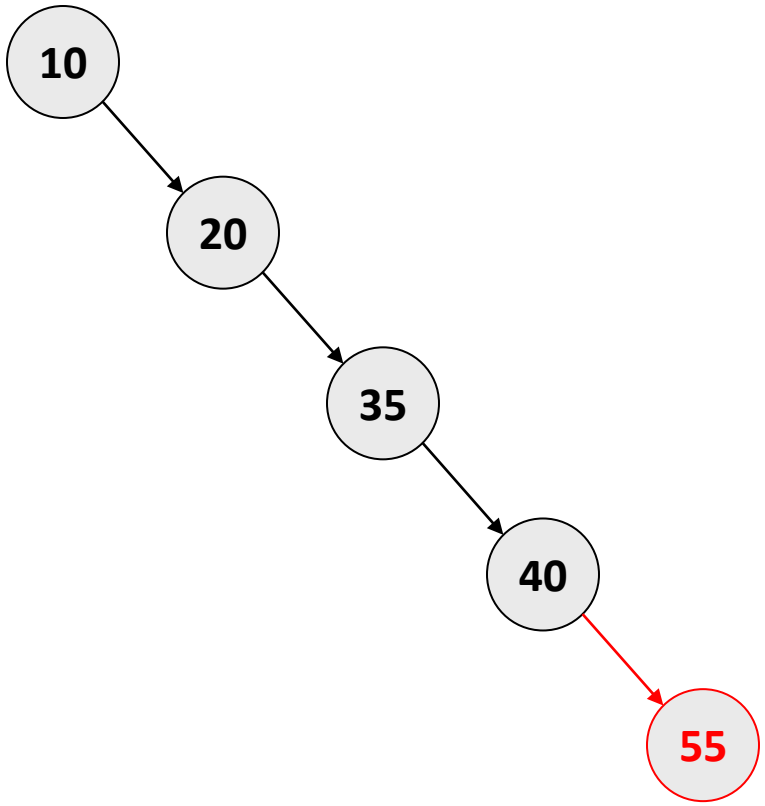
Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;



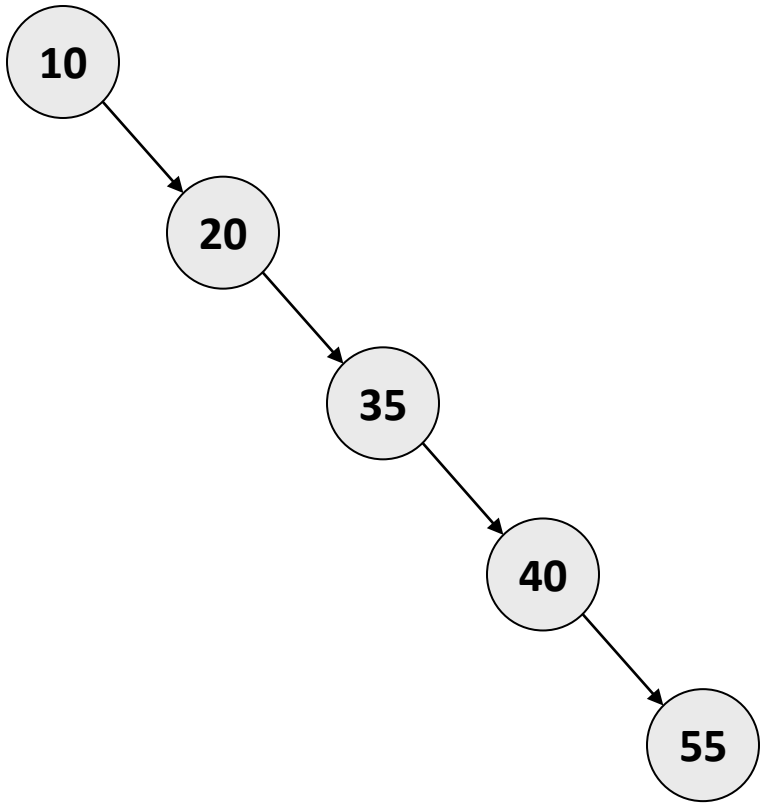
Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;



Problemas de balanceamento

- Exemplo: inserção dos valores 10, 20, 35, 40, 55;



Árvore Degenerada!
(*Lista Encadeada*)

Problemas de balanceamento

- A manipulação de uma ABB pode levar ao seu desbalanceamento!

Problemas de balanceamento

- A manipulação de uma ABB pode levar ao seu desbalanceamento!

- O caso extremo é quando a árvore se torna **degenerada** (equivalente a uma *lista encadeada*);

Problemas de balanceamento

- A manipulação de uma ABB pode levar ao seu desbalanceamento!

- O caso extremo é quando a árvore se torna **degenerada** (equivalente a uma *lista encadeada*);

- Neste caso, o desempenho dos algortimos cai, tendendo à ordem linear → *O(n)*.

Solução: ABB balanceadas

- Para lidar com este problema, existem algumas implementações de **ABBs autobalanceáveis**;

Solução: ABB balanceadas

- Para lidar com este problema, existem algumas implementações de **ABBs autobalanceáveis**;
  - Organizam sua estrutura a cada manipulação;

Solução: ABB balanceadas

- Para lidar com este problema, existem algumas implementações de **ABBs autobalanceáveis**;
  - Organizam sua estrutura a cada manipulação;
  - Garantem que a altura da ABB seja sempre *O(log n)*;

Solução: ABB balanceadas

- Para lidar com este problema, existem algumas implementações de **ABBs autobalanceáveis**;
  - Organizam sua estrutura a cada manipulação;
  - Garantem que a altura da ABB seja sempre *O(log n)*;

- Exemplos:
  - Árvores AVL;
  - Árvores Bicolores (red-black trees);
  - *Splay Trees*;

Próximo tópico de estudo

- **Árvores AVL**!