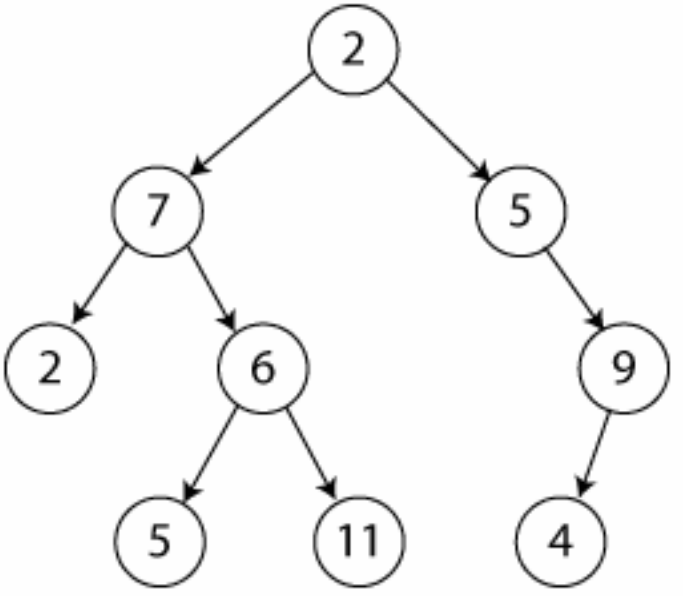
ÁRVORES BINÁRIAS DE BUSCA

Prof. Muriel Mazzetto

Estrutura de Dados

Árvore binária

- Uma estrutura de dados do tipo árvore **ordenada.**
- Cada nó indica o filho à **esquerda** e o filho à **direita.**

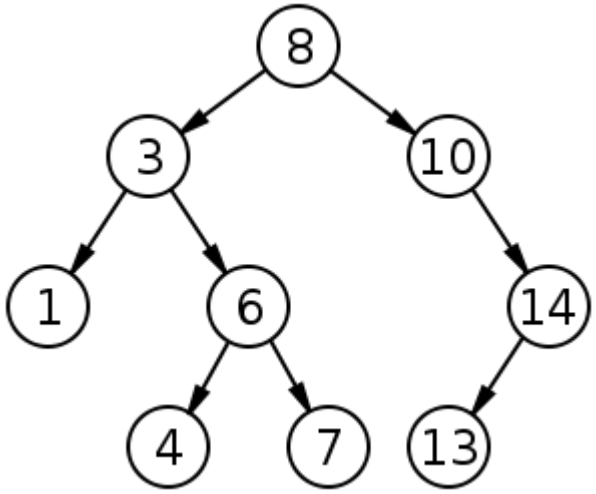


Árvore binária de busca

- Mantém as características de uma árvore binária.
- Possui restrições de inserção e remoção que deixam dados ordenados.
  - Quanto mais à esquerda, menor o valor.
  - Quanto mais à direita, maior o valor.
- Aplicação:
  - Busca binária.
  - Ordenação de dados.

Árvore binária de busca

- Para cada nó *pai*:
  - Todos os valores (*chaves*) da **subárvore esquerda são menores** que o nó pai.
  - Todos os valores (*chaves*) da **subárvore direita são maiores** que o nó pai.



Árvore binária de busca

- Operações em uma ABB:
  - Criação da árvore.
  - Inserção.
  - Remoção.
  - Busca.
  - Destruição da árvore.

Tipos de dados da ABB


```c
//Arquivo .c
#include <stdio.h>
#include <stdlib.h>
#include "ArvoreBinaria.h"

struct NO{
    int info;
    struct NO *esq;
    struct NO *dir;
};
```

```c
//Arquivo .h
typedef struct NO* ArvBin;
```

```c
//Arquivo main.c
ArvBin* raiz = cria_ArvBin();
```

Criação da ABB

- Criar a raiz da árvore.
- É um ponteiro de ponteiro para os elementos (similar às listas encadeadas, filas e pilhas).

```
ArvBin* cria_ArvBin(){
    ArvBin* raiz = (ArvBin*) malloc(sizeof(ArvBin));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}
```

Destruição da ABB

- Percorrer todos os nós da árvore, para liberar a memória alocada dinamicamente.

```
void libera_NO(struct NO* no){
    if(no == NULL)
        return;
    libera_NO(no->esq);
    libera_NO(no->dir);
    free(no);
    no = NULL;
}

void libera_ArvBin(ArvBin* raiz){
    if(raiz == NULL)
        return;
    libera_NO(*raiz);//libera cada nó
    free(raiz);//libera a raiz
}
```

Percurso na ABB

- Percorrer todos os nós da árvore, visitando cada um uma única vez.

```
void preOrdem_ArvBin(ArvBin *raiz){
    if(raiz == NULL)
        return;
    if(*raiz != NULL){
        printf("%d\n",(*raiz)->info);
        preOrdem_ArvBin(&((*raiz)->esq));
        preOrdem_ArvBin(&((*raiz)->dir));
    }
}
```

Inserção em ABB

- Receber a raiz de uma árvore e um dado.
- Encontrar o local vazio (folha) que mantenha a ordem das chaves.
  - Verificar se a chave é menor que a raiz:
    - Descer para a subárvore esquerda;
  - Verificar se a chave é maior que a raiz:
    - Descer para a subárvore direita;
- Atualizar os ponteiros para inserir o novo nó.
- Pode ser feito recursivamente ou iterativamente.

Inserção em ABB

- Algoritmo insere(*raiz, dado):
  - Se posição está vazia (NULL), insere o dado;
  - Se é menor que raiz, insere((*raiz)->**esquerda**, **dado**);
  - Se é maior que raiz, insere((*raiz)->**direita**, **dado**);
  - Se é igual, descarta o dado.

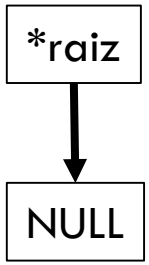
Inserção em ABB

- Algoritmo insere(*raiz, dado):
  - Se posição está vazia (NULL), insere o dado;
  - Se é menor que raiz, insere((*raiz)->**esquerda**, **dado**);
  - Se é maior que raiz, insere((*raiz)->**direita**, **dado**);
  - Se é igual, descarta o dado.

- 1) Insira os seguintes elementos em uma ABB:
  - 4, 14, 8, 3, 15, 16, 2, 23, 42

Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

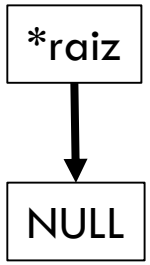


4

Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

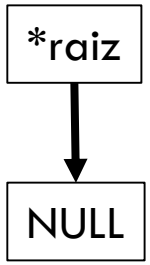
4



Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

4

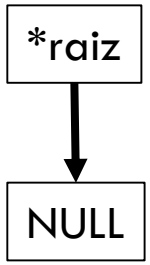


novo

Inserção em ABB

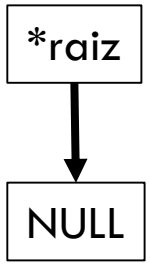

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

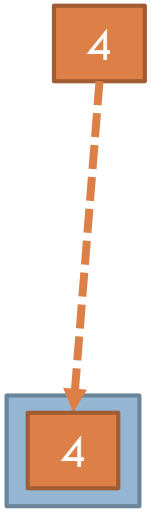
4



Inserção em ABB

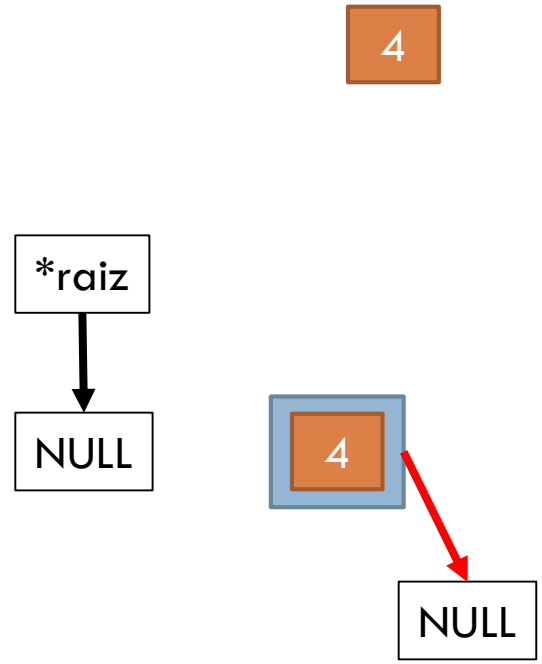

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```





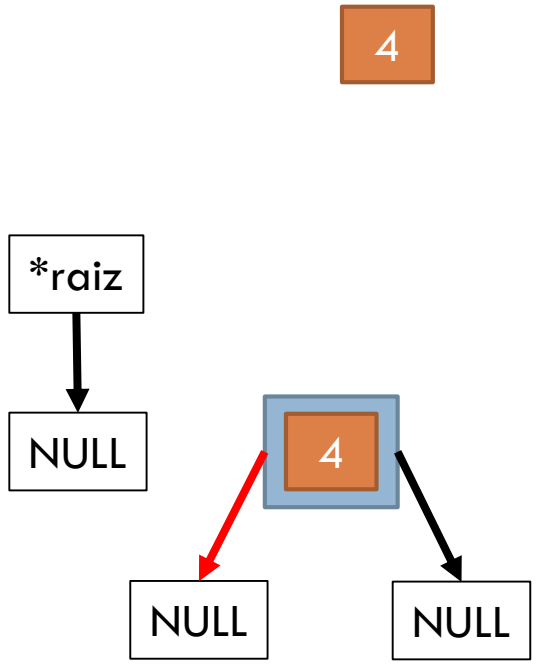
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



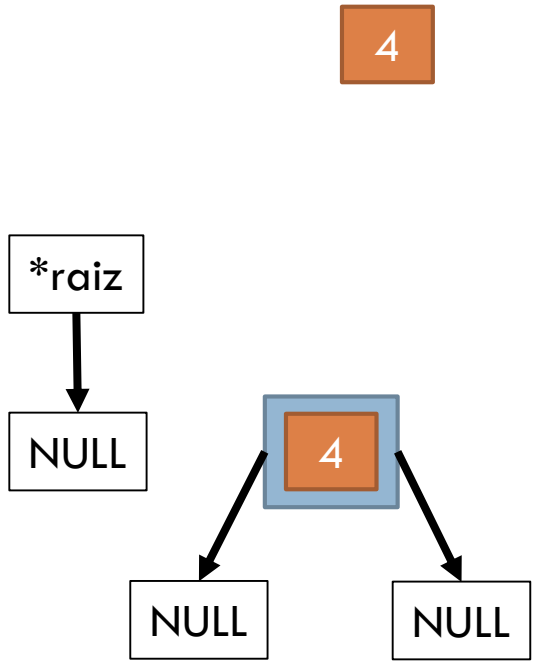
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



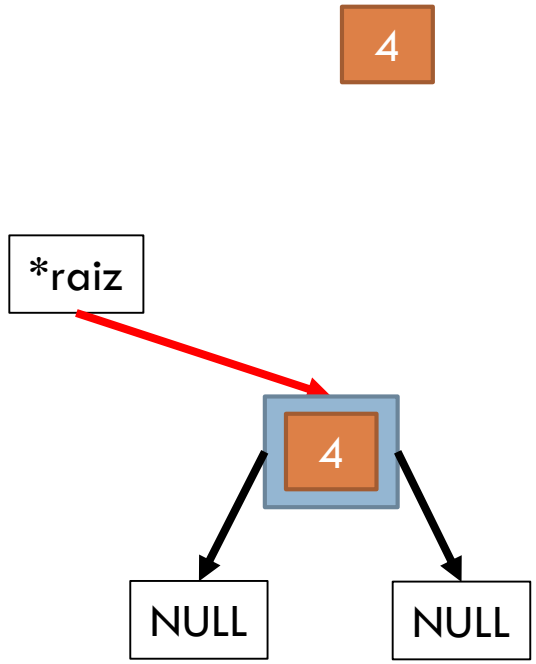
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



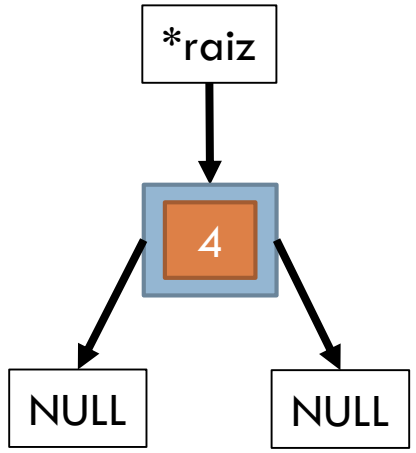
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
     →  *raiz = novo;
          [...]
```

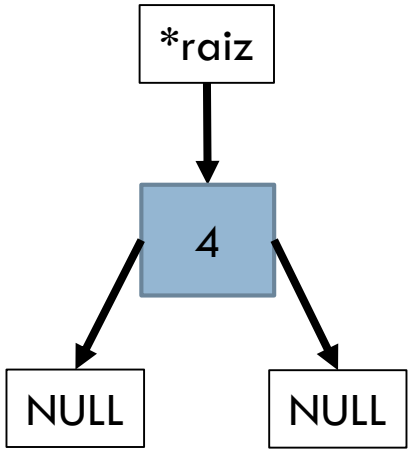


Inserção em ABB


```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



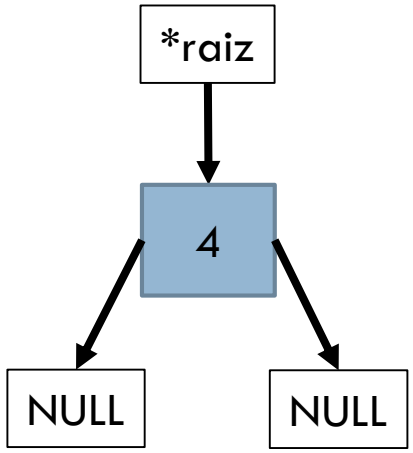
Inserção em ABB



Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

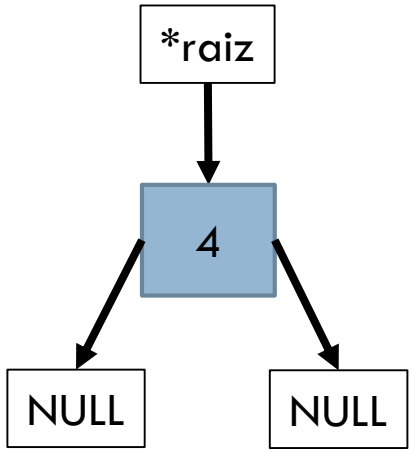
9



Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

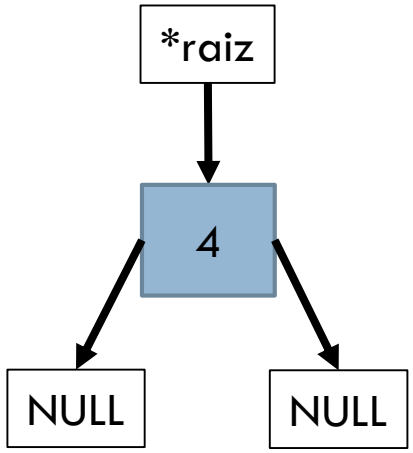
9



Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

9

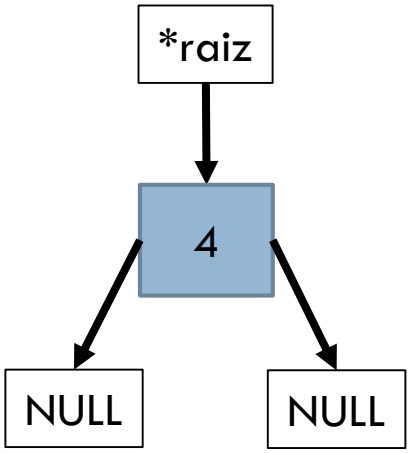


novo

Inserção em ABB

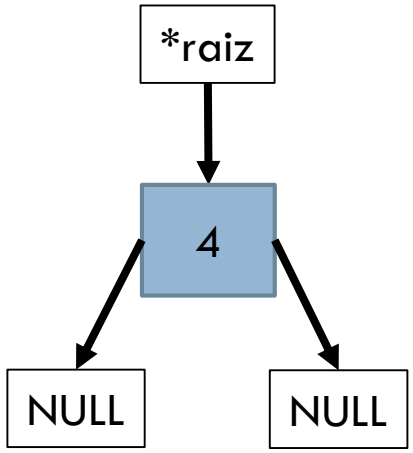

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

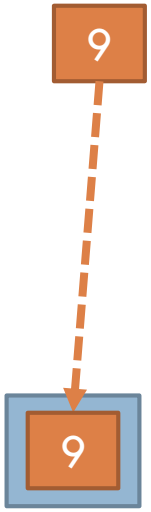
9



Inserção em ABB

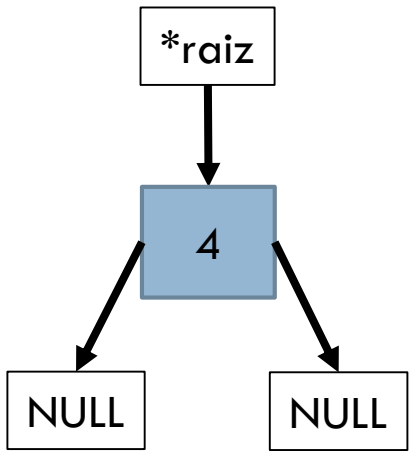

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

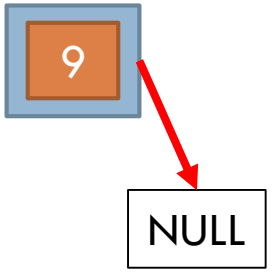




Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

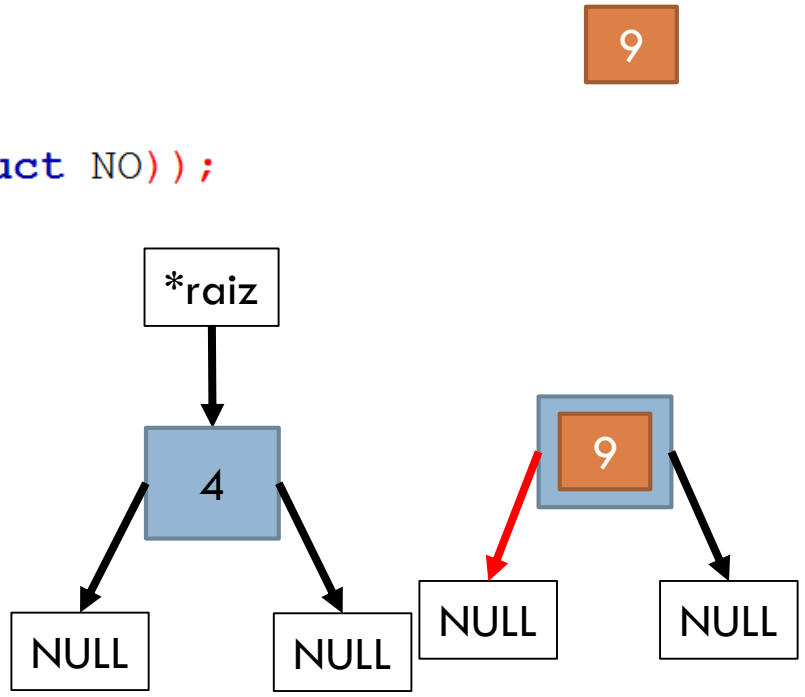




9

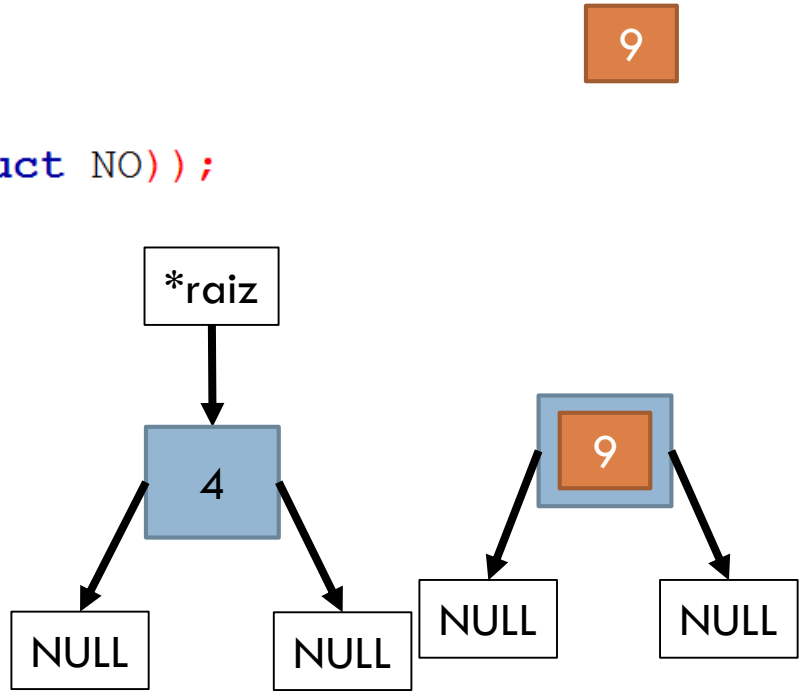
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



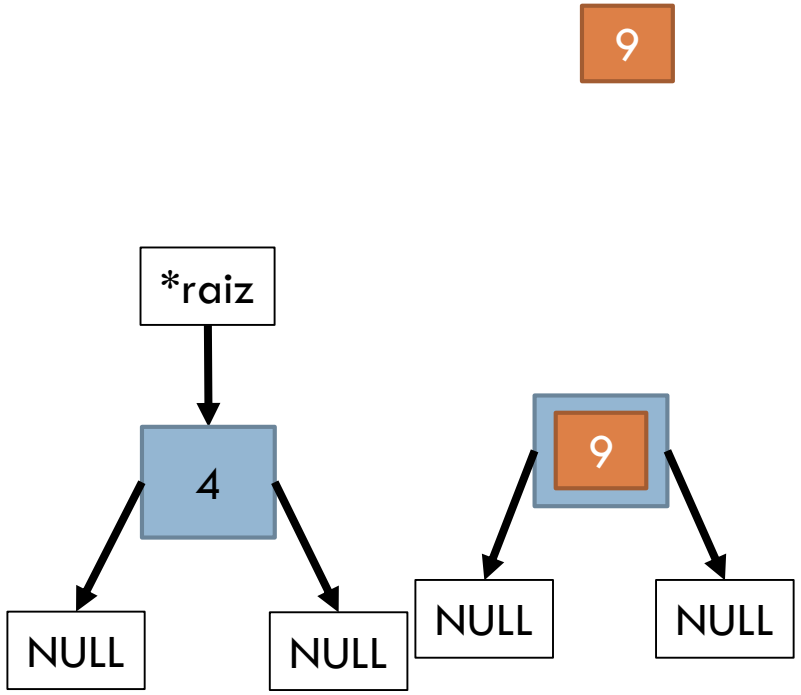
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



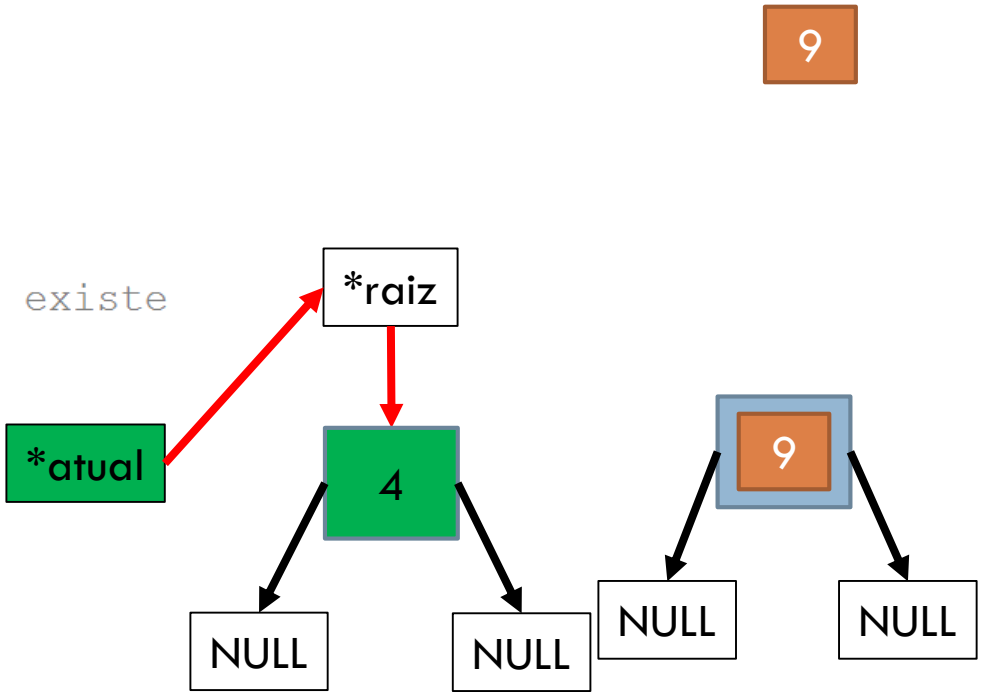
Inserção em ABB


```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



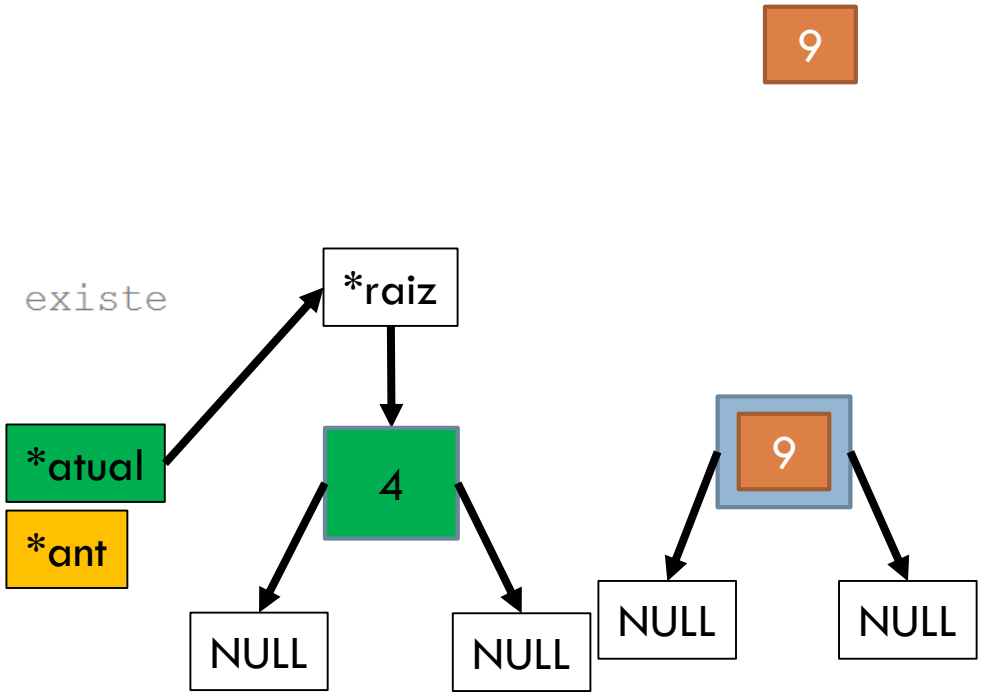
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



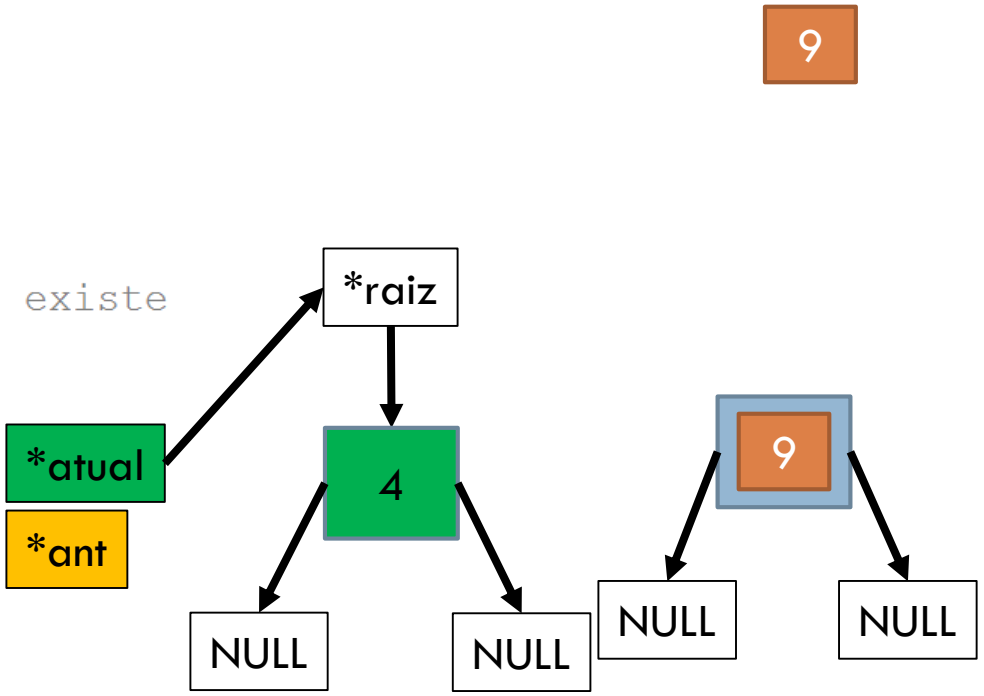
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



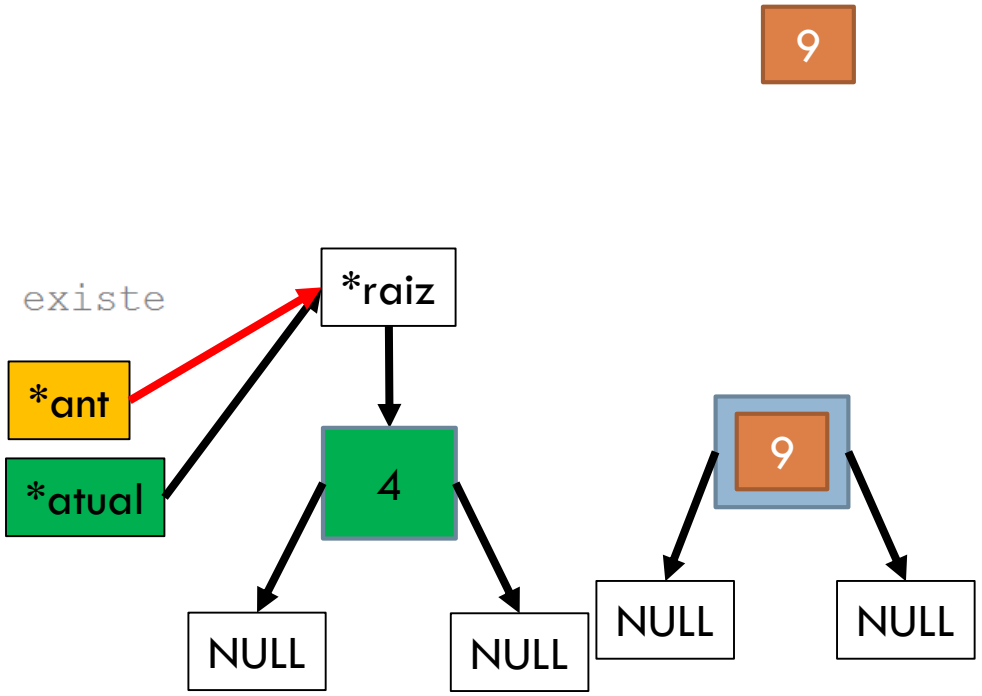
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



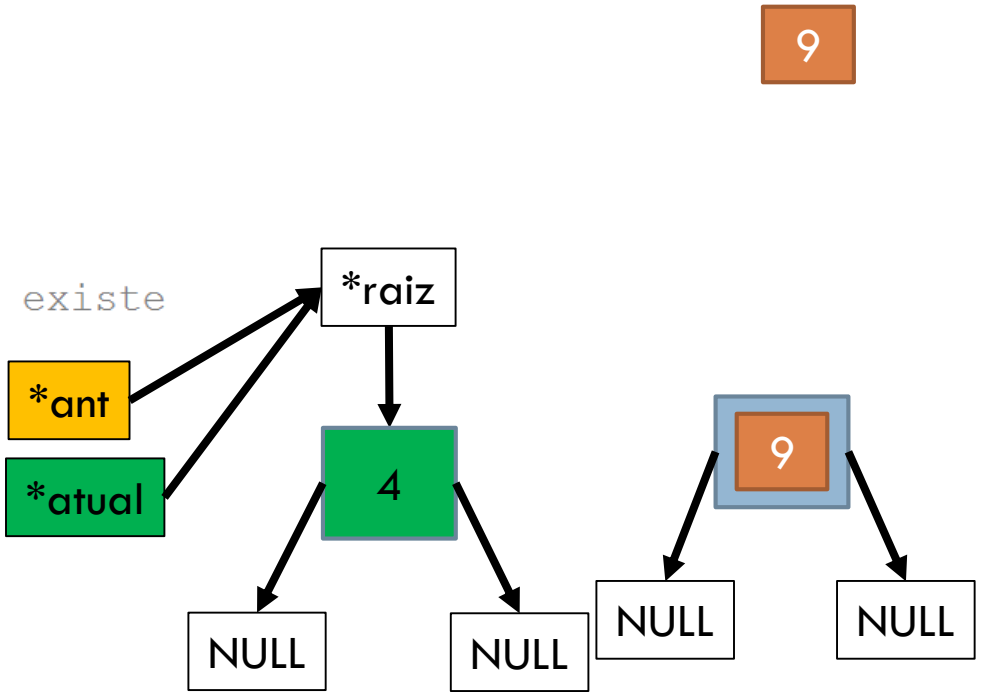
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
    ➡ ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



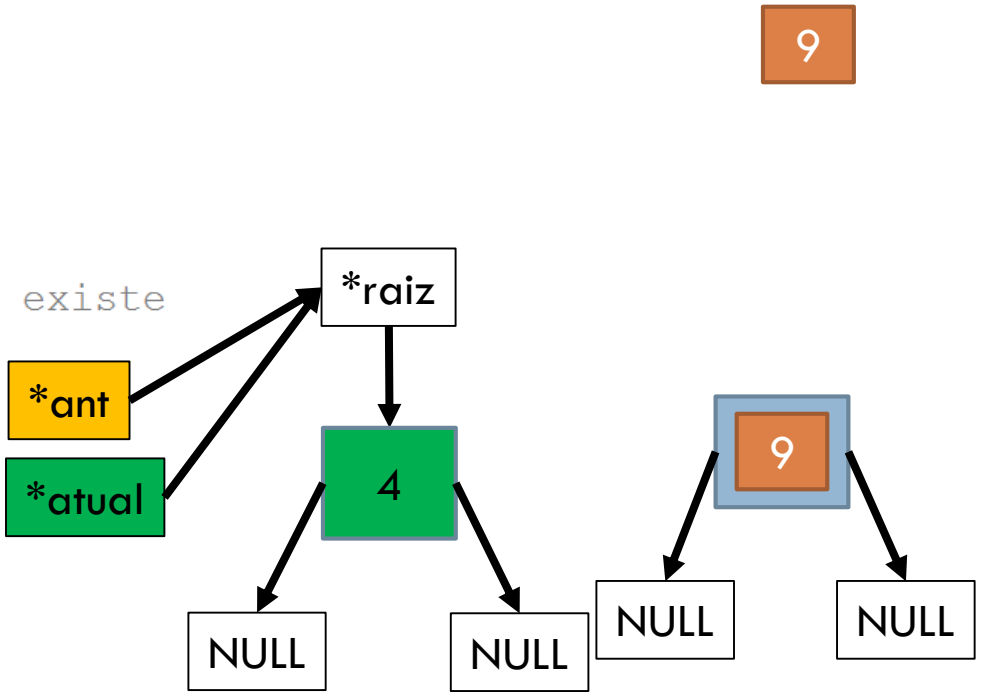
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



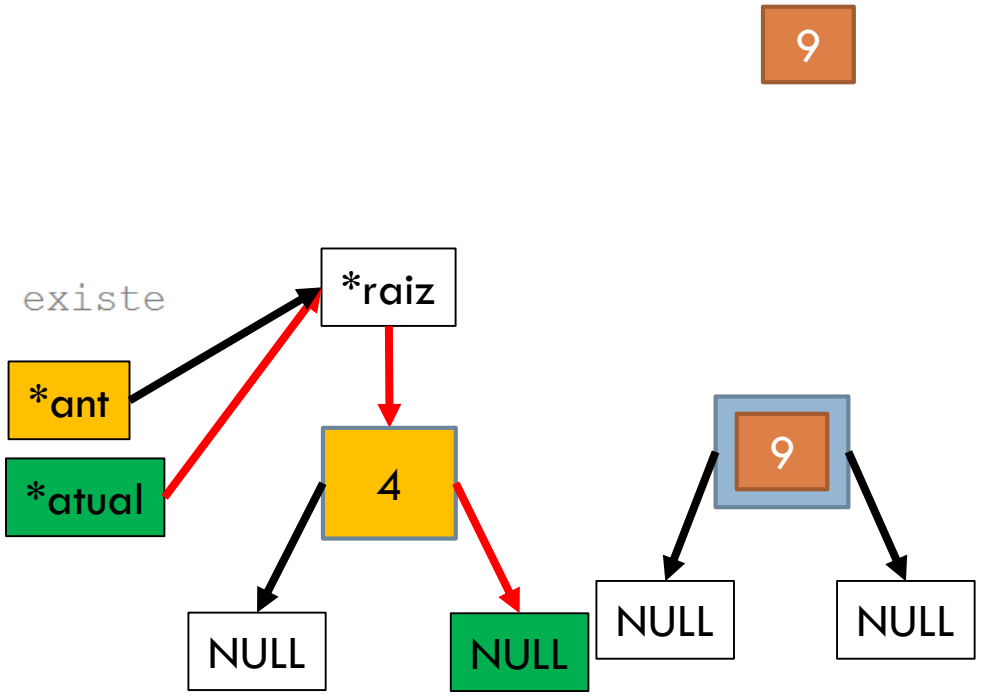
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



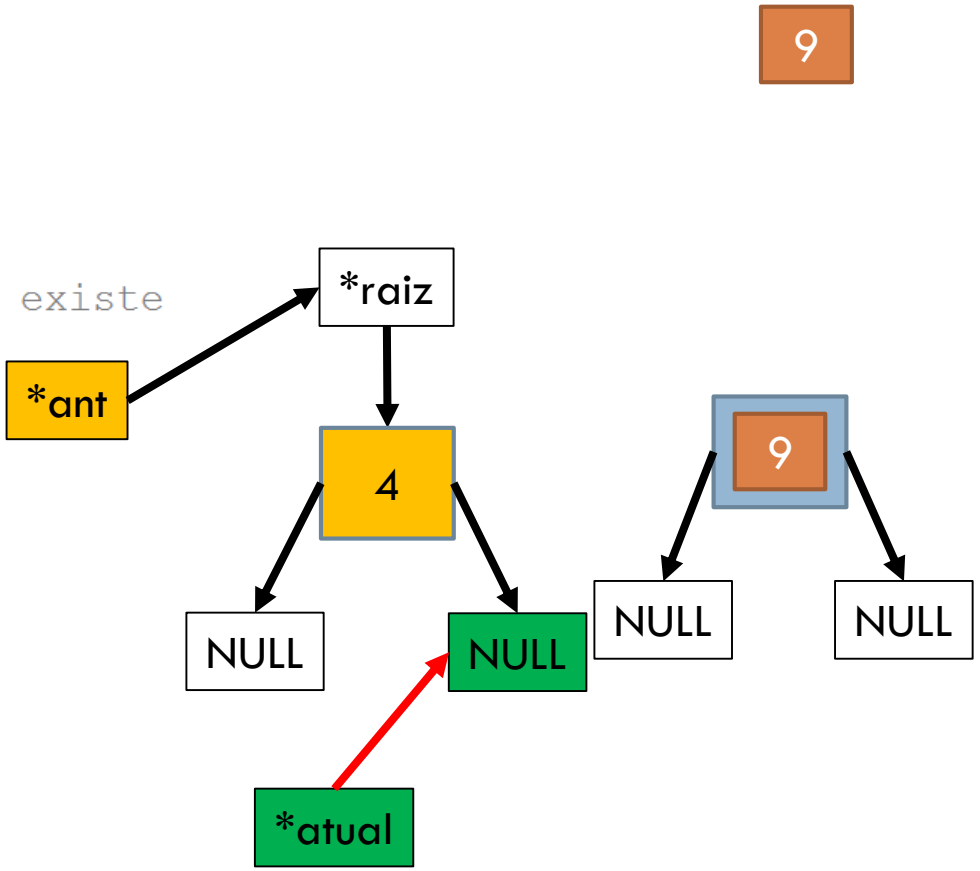
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



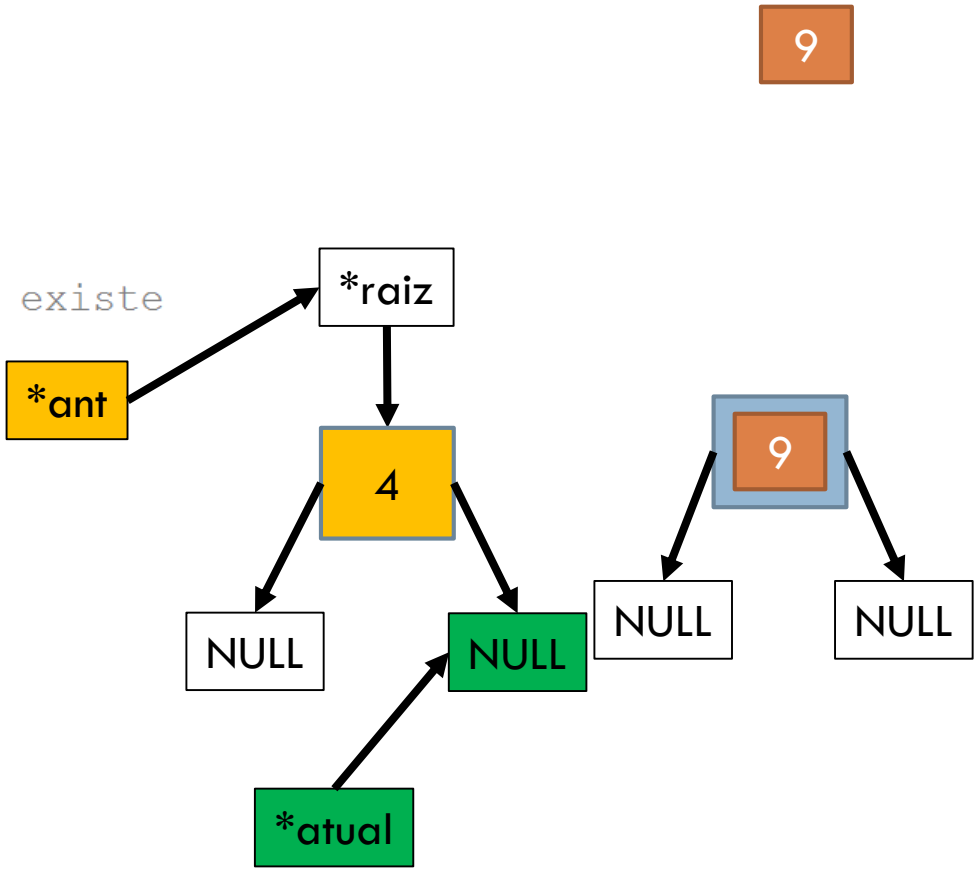
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



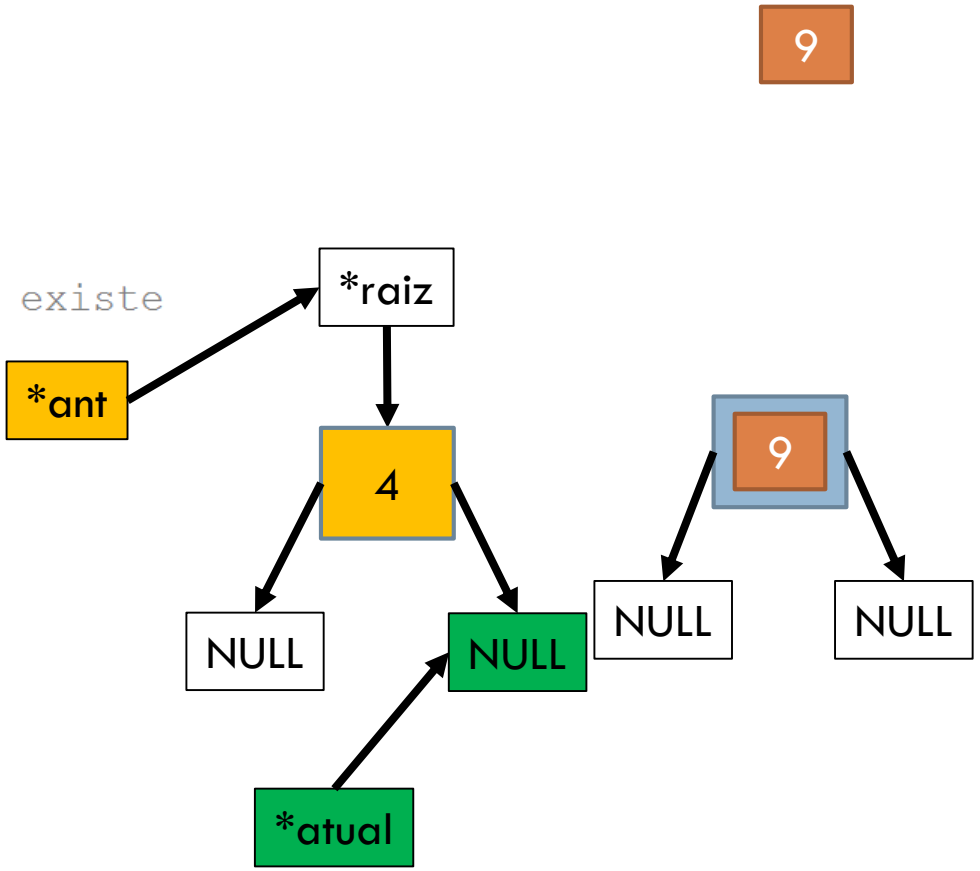
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



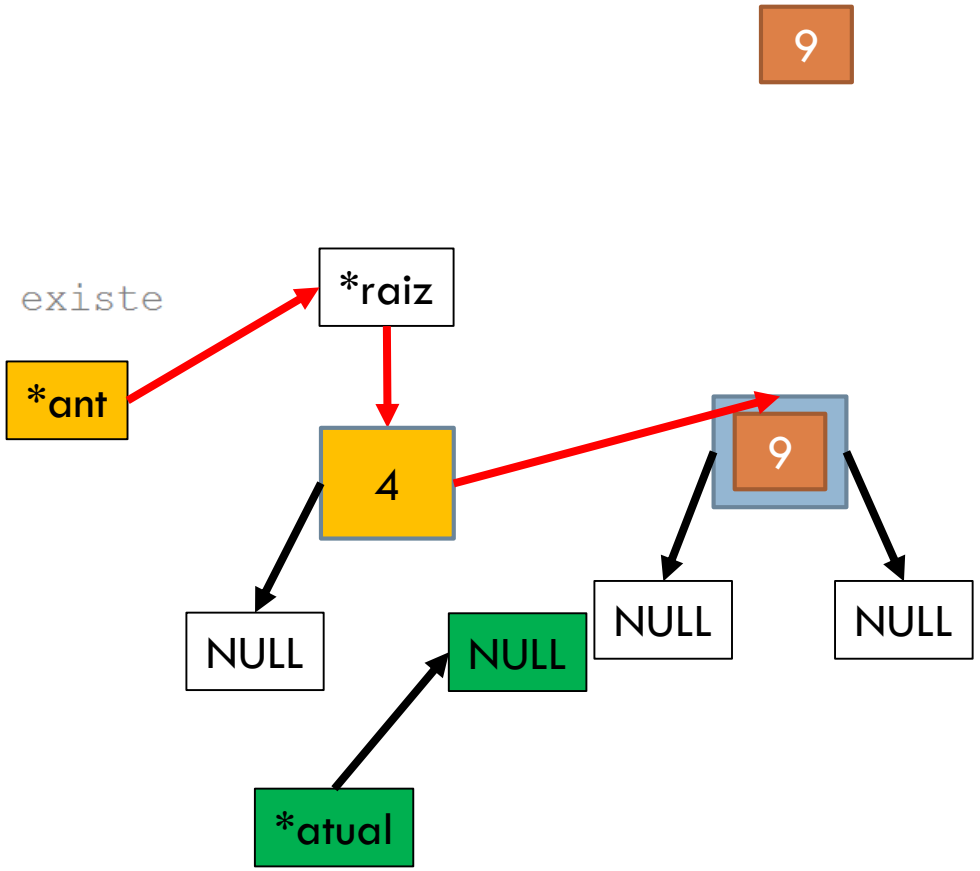
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

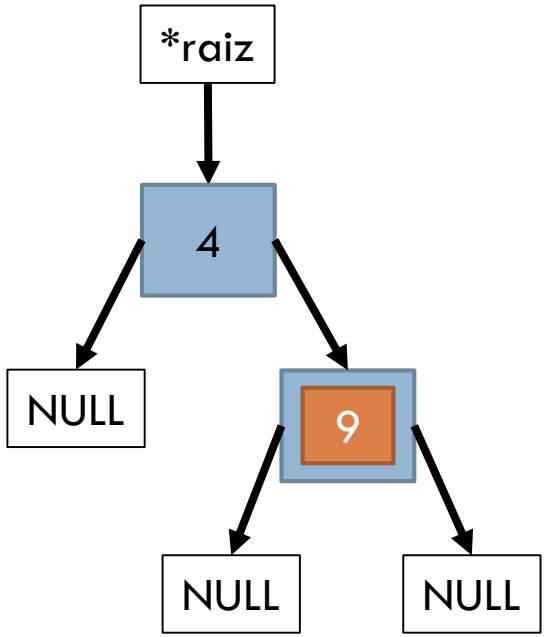

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
    ➡ ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



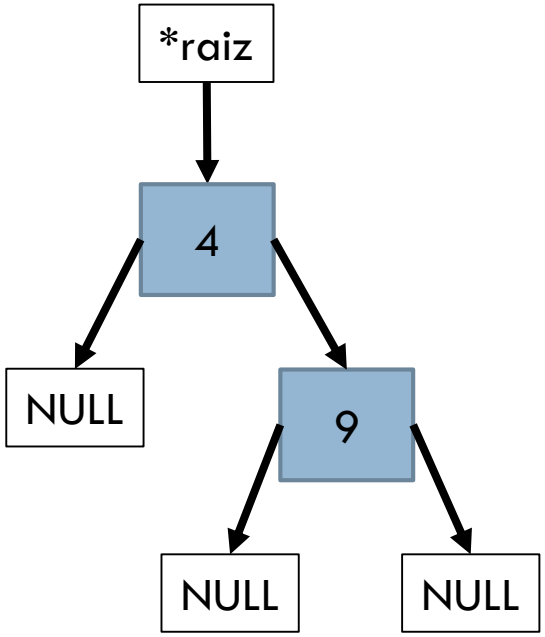
Inserção em ABB


```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

9

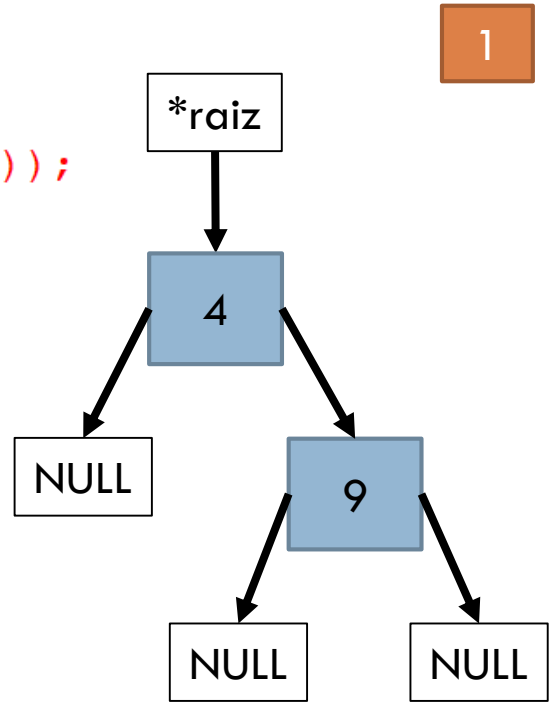


Inserção em ABB



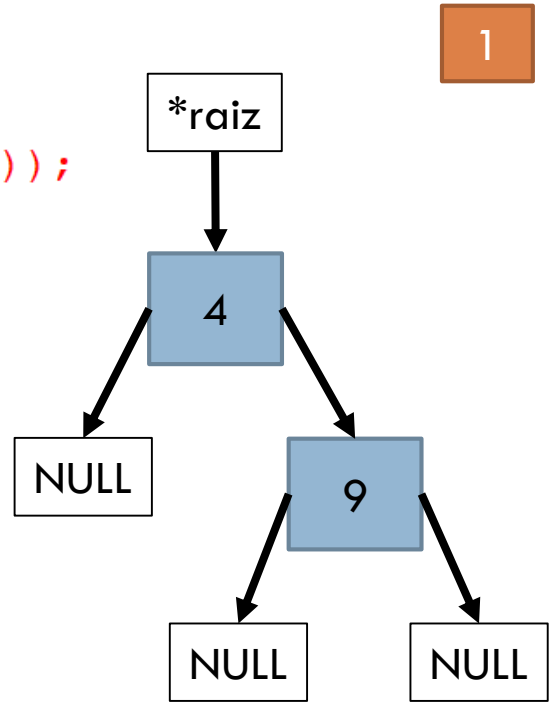
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



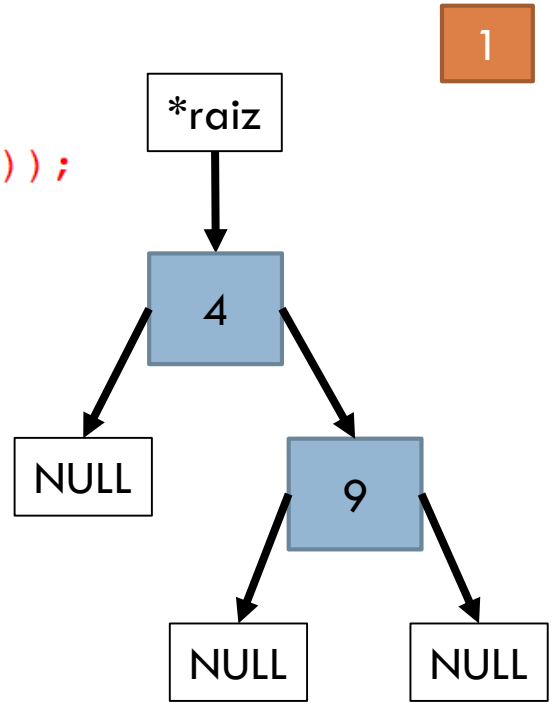
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



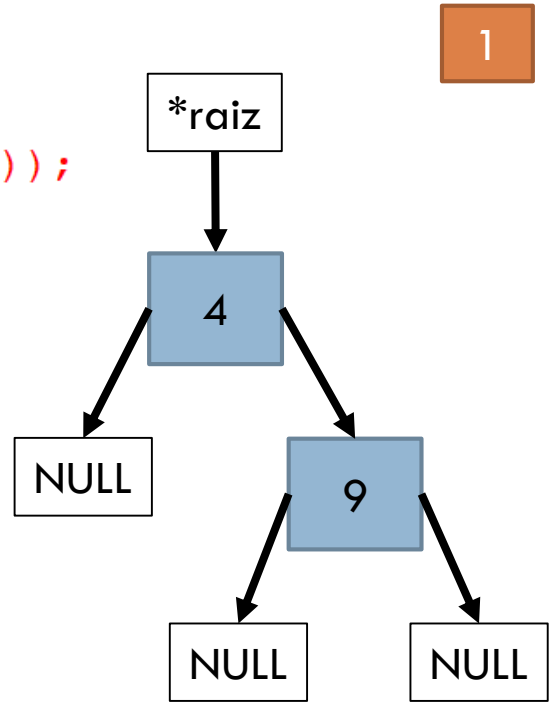
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



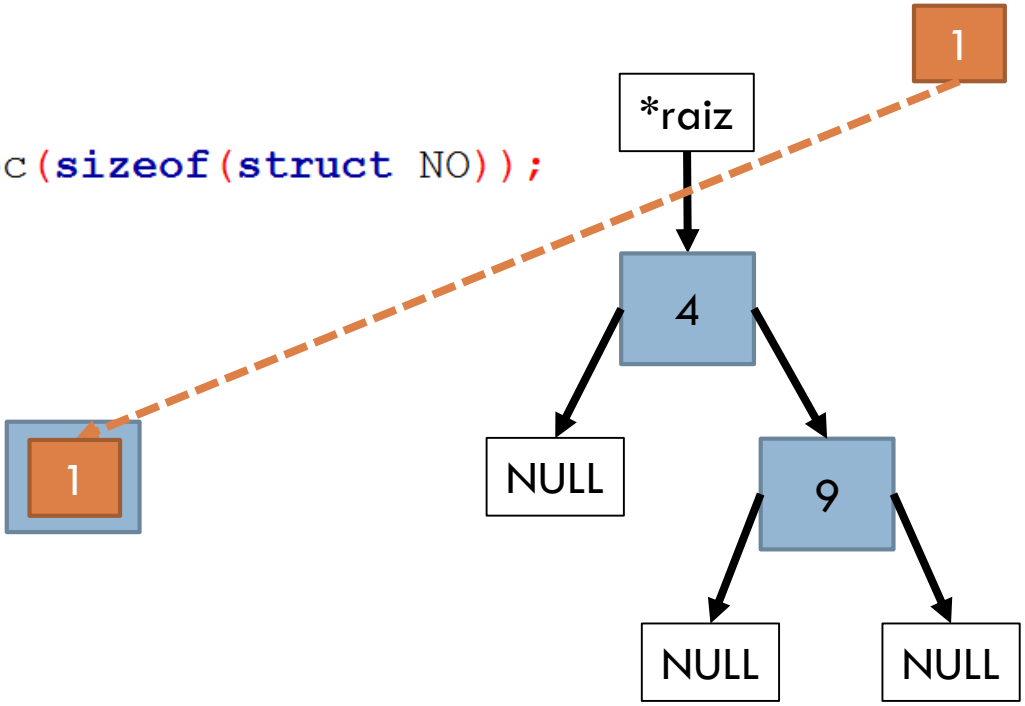
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



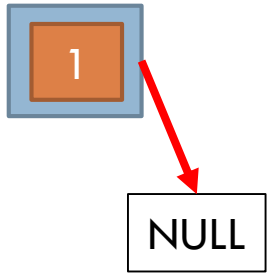
Inserção em ABB

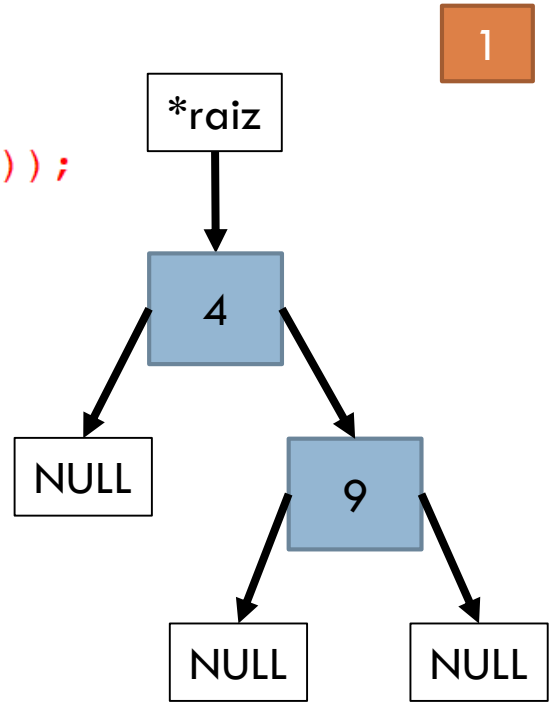

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



Inserção em ABB

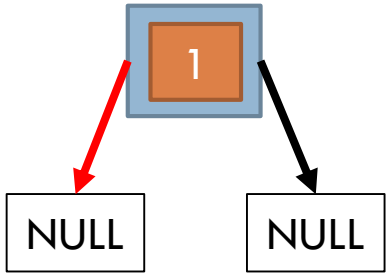

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

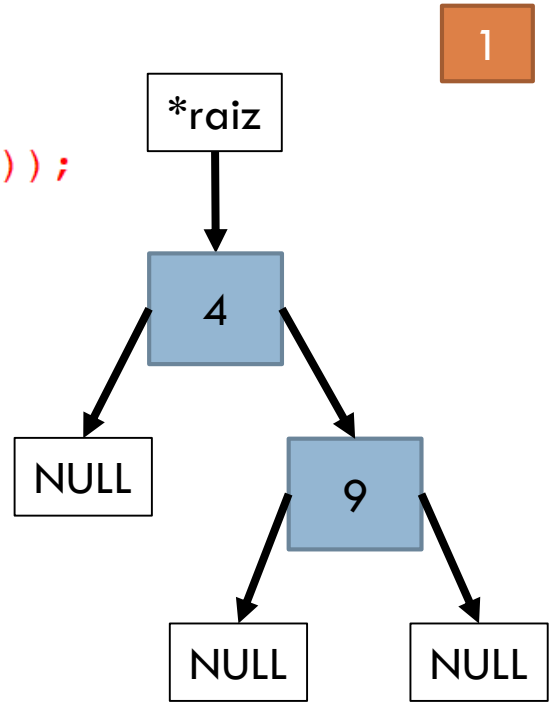




Inserção em ABB

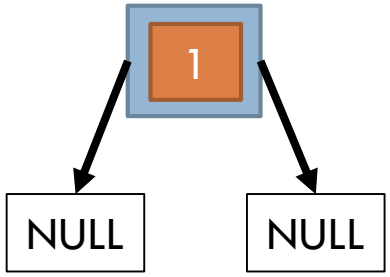

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

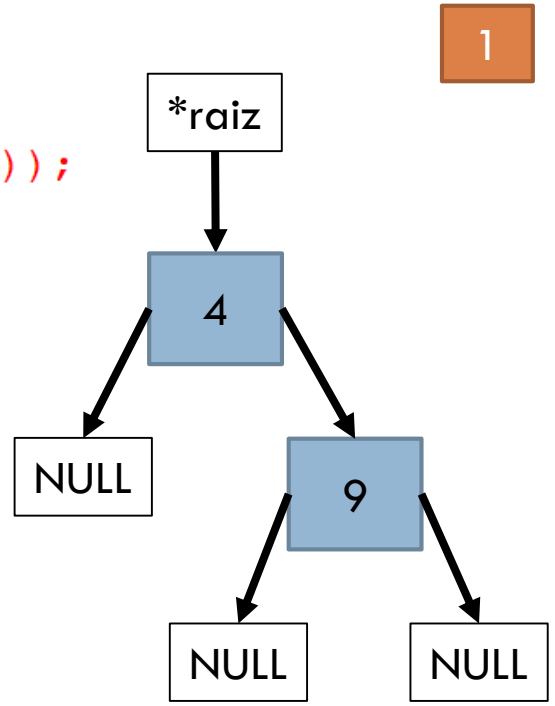




Inserção em ABB

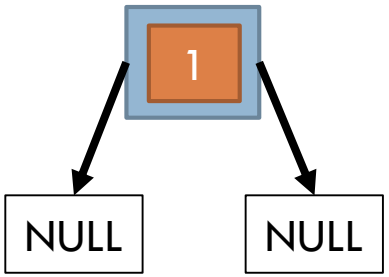

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
        [...]
```

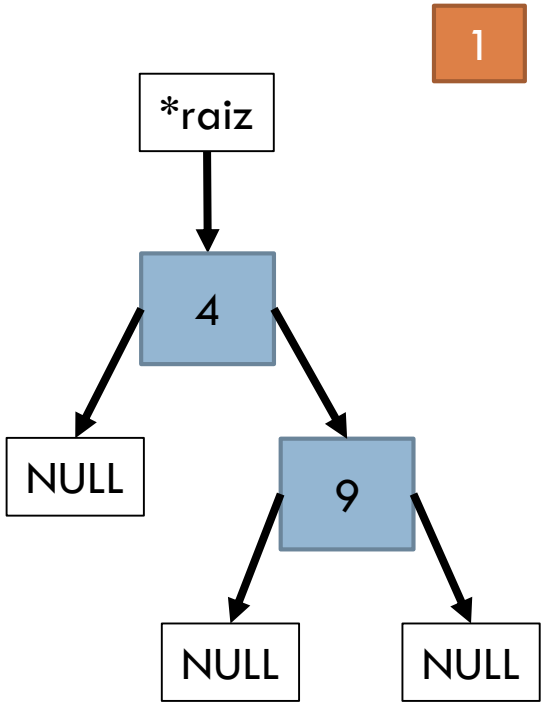




Inserção em ABB

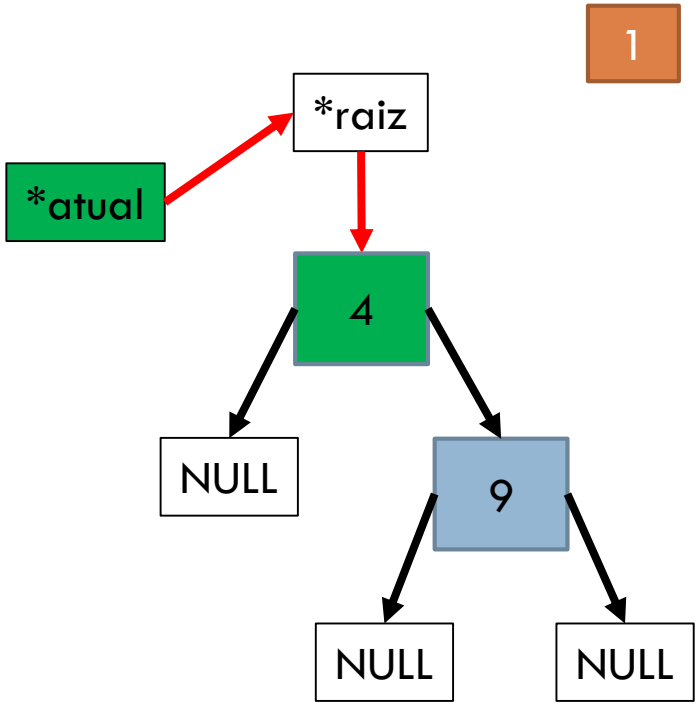

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

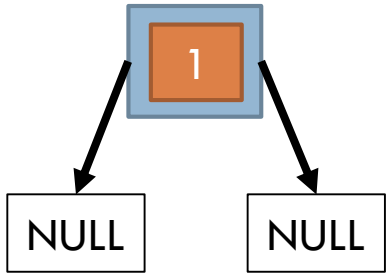




Inserção em ABB

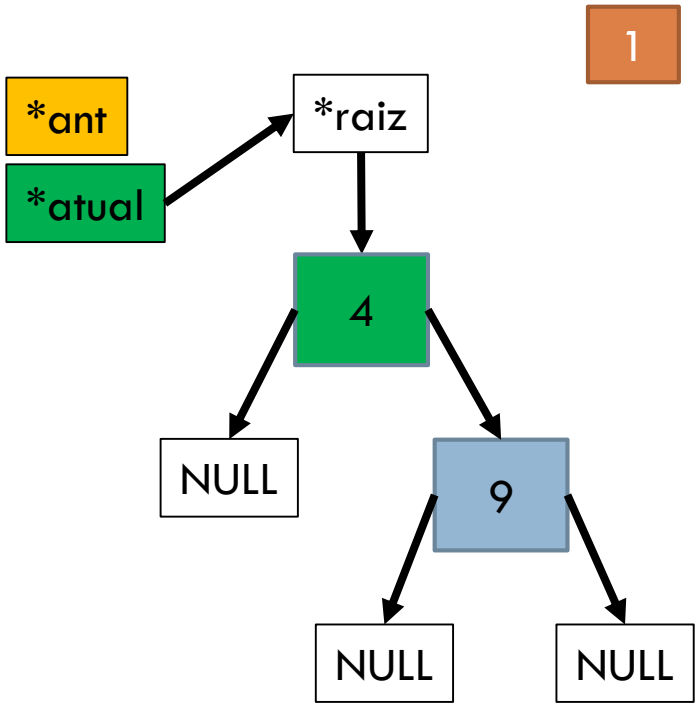

```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

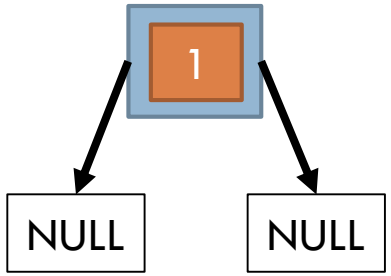




Inserção em ABB

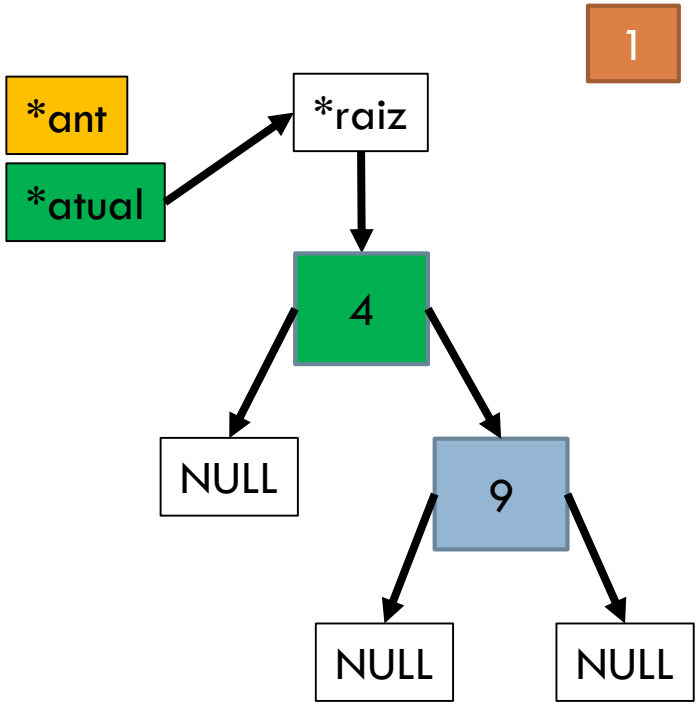

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

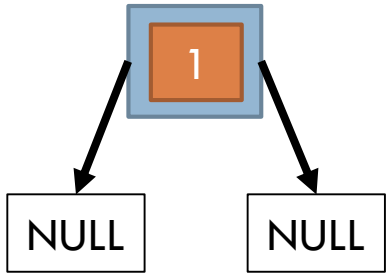




Inserção em ABB

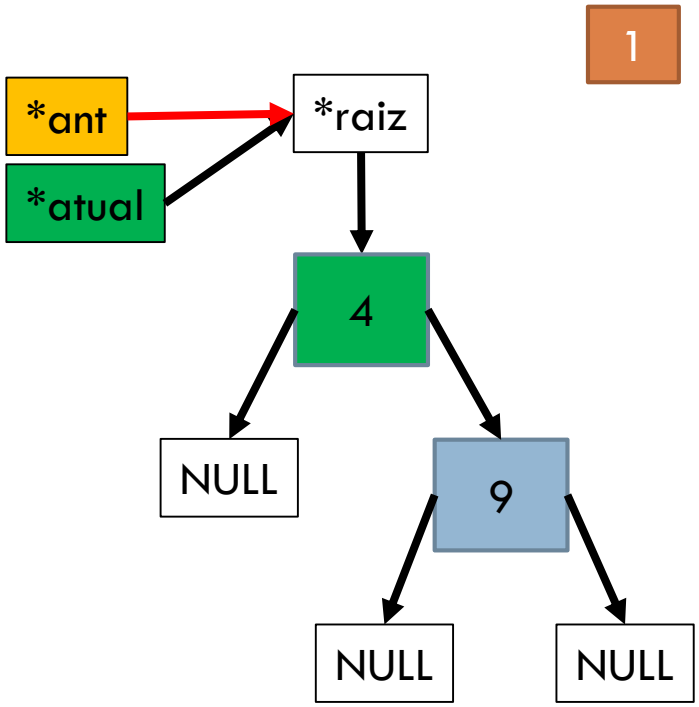

```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

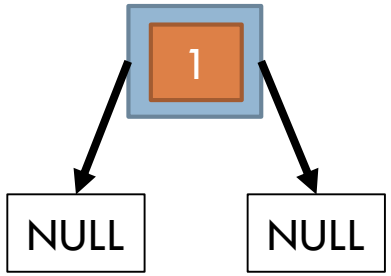




Inserção em ABB

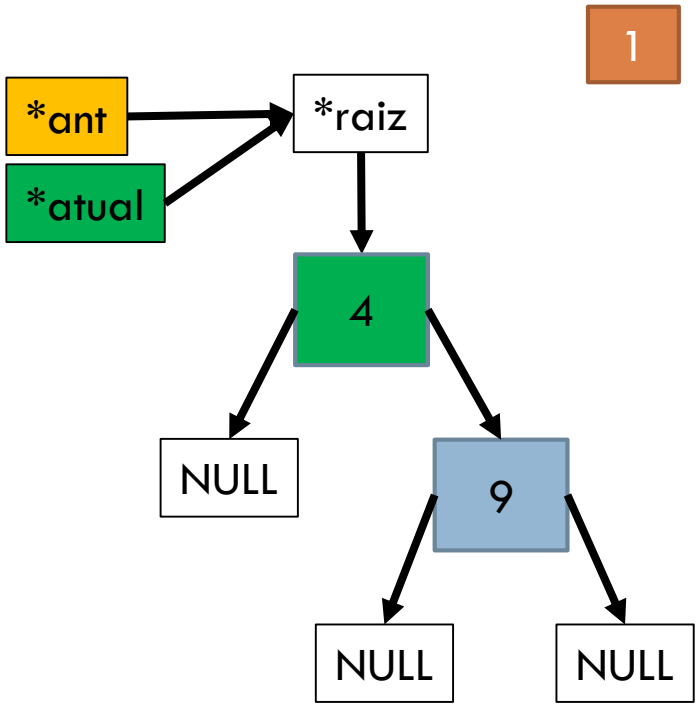

```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

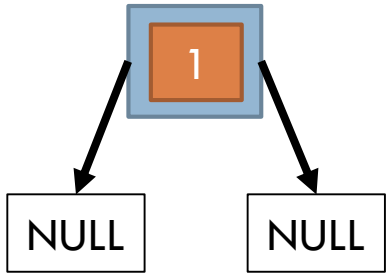




Inserção em ABB

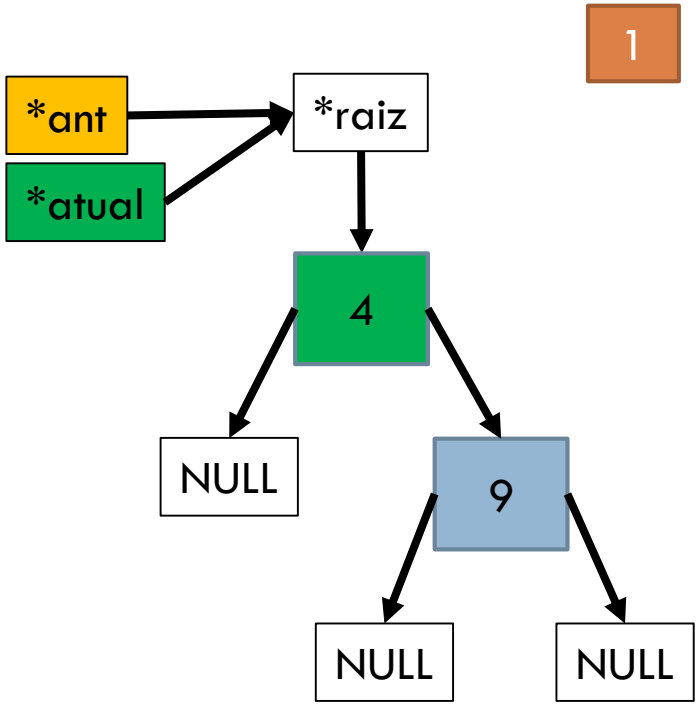

```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

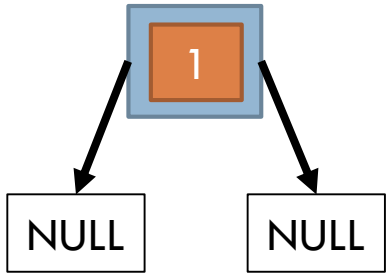




Inserção em ABB

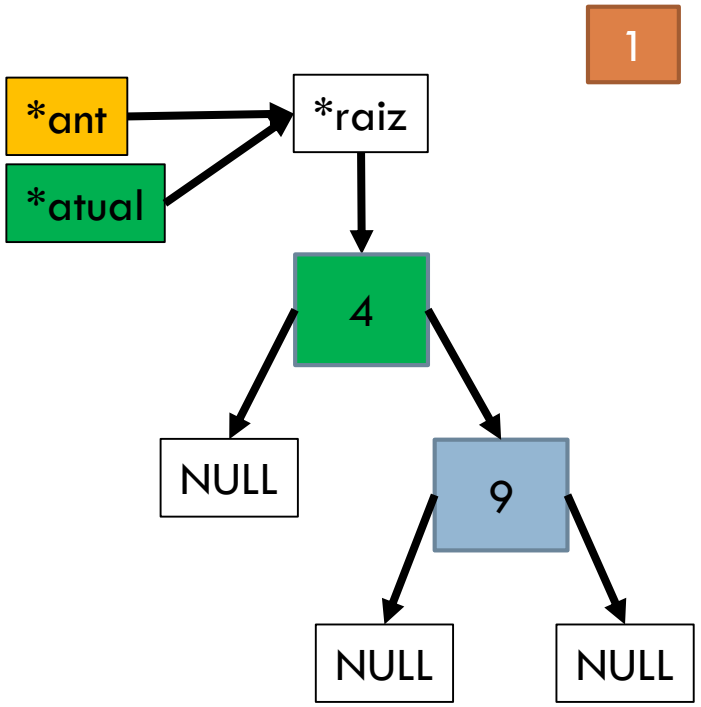

```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

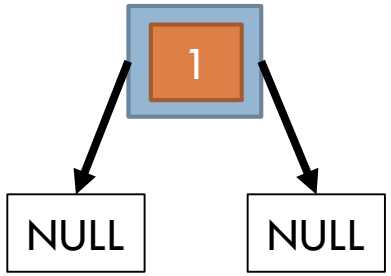




Inserção em ABB

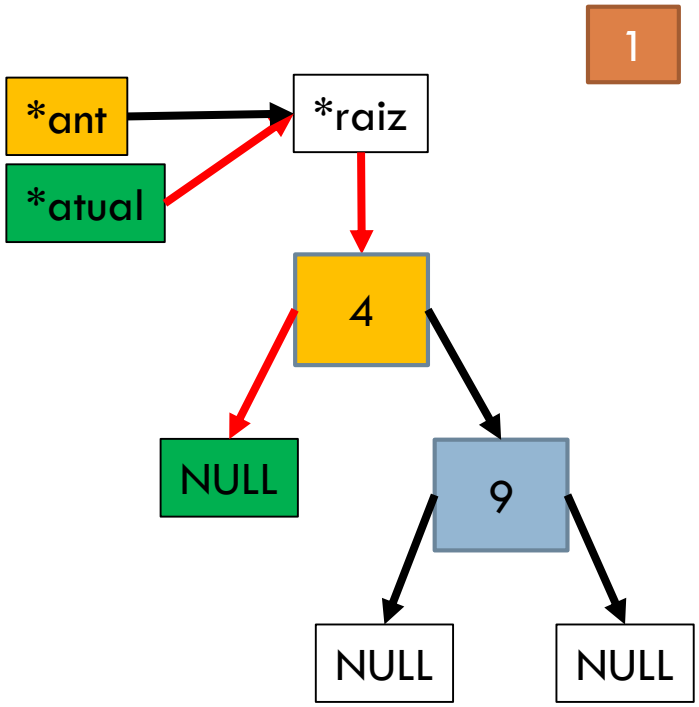

```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

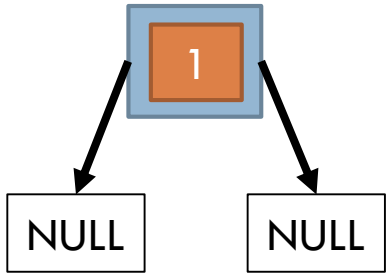




Inserção em ABB

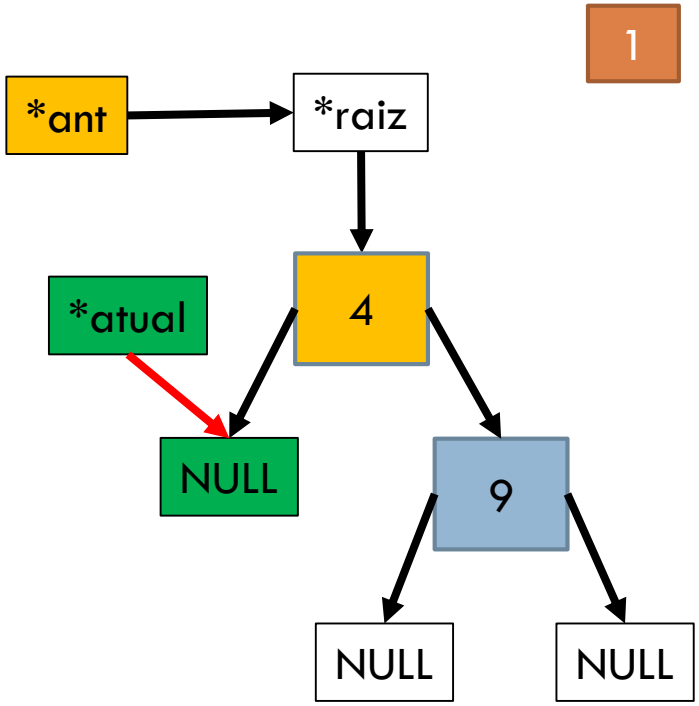

```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

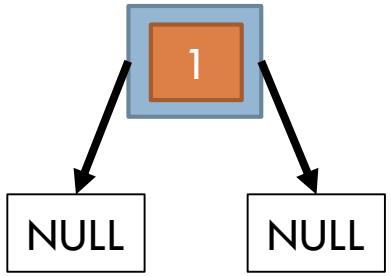




Inserção em ABB

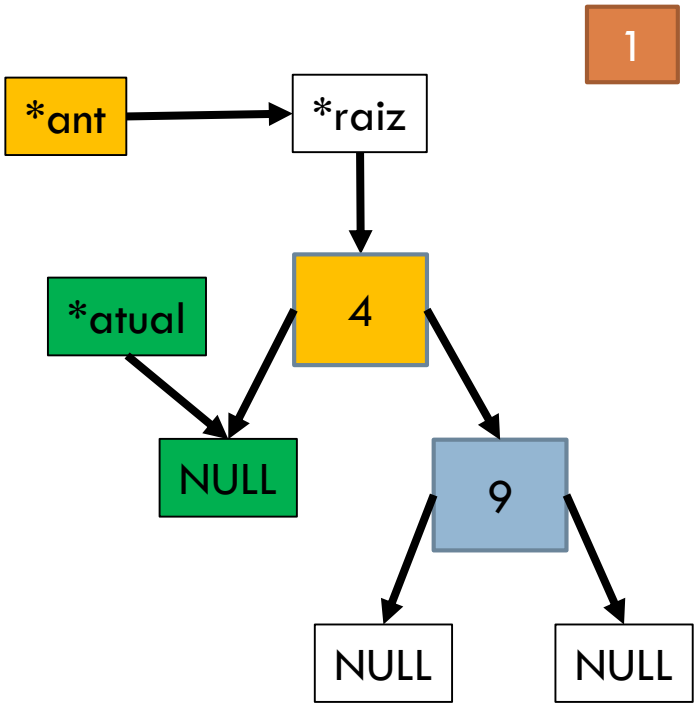

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

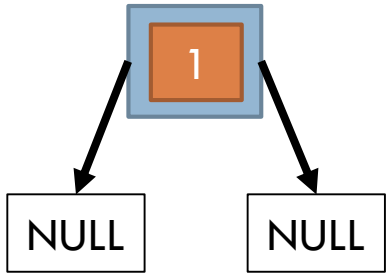




Inserção em ABB

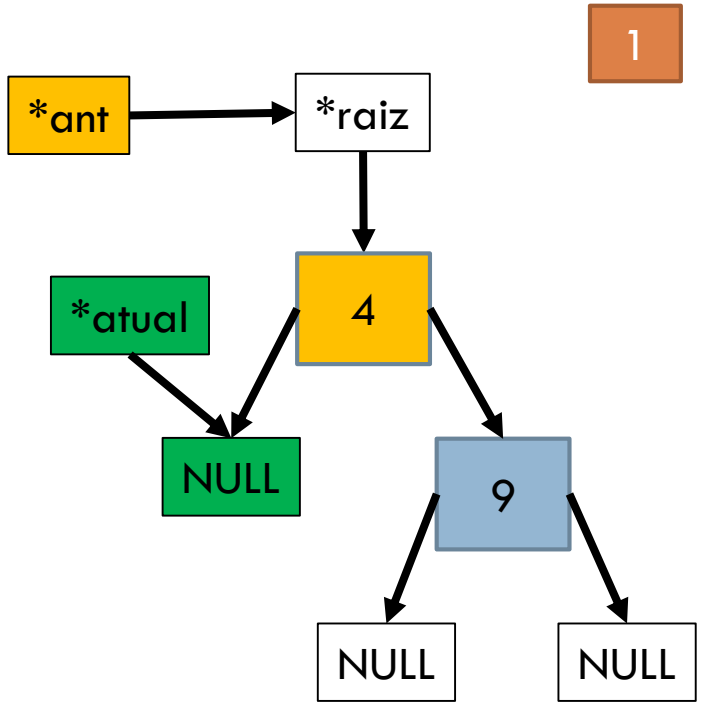

```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

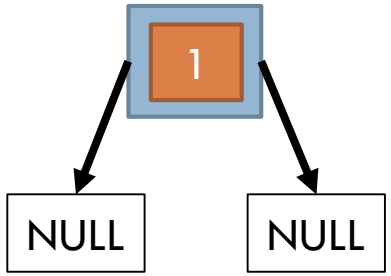




Inserção em ABB

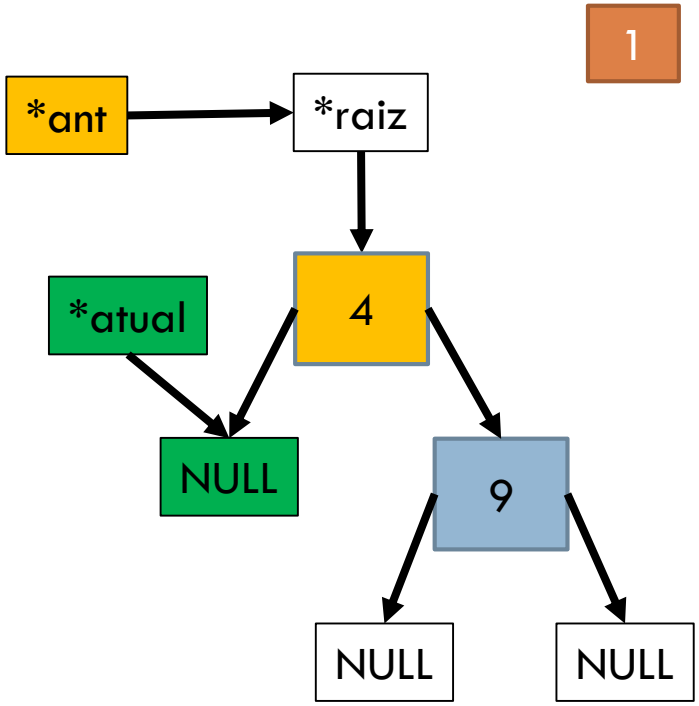

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

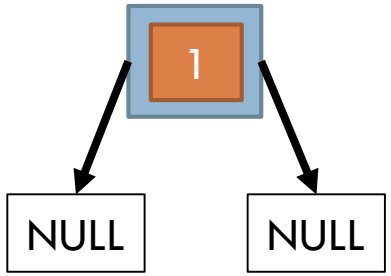




Inserção em ABB

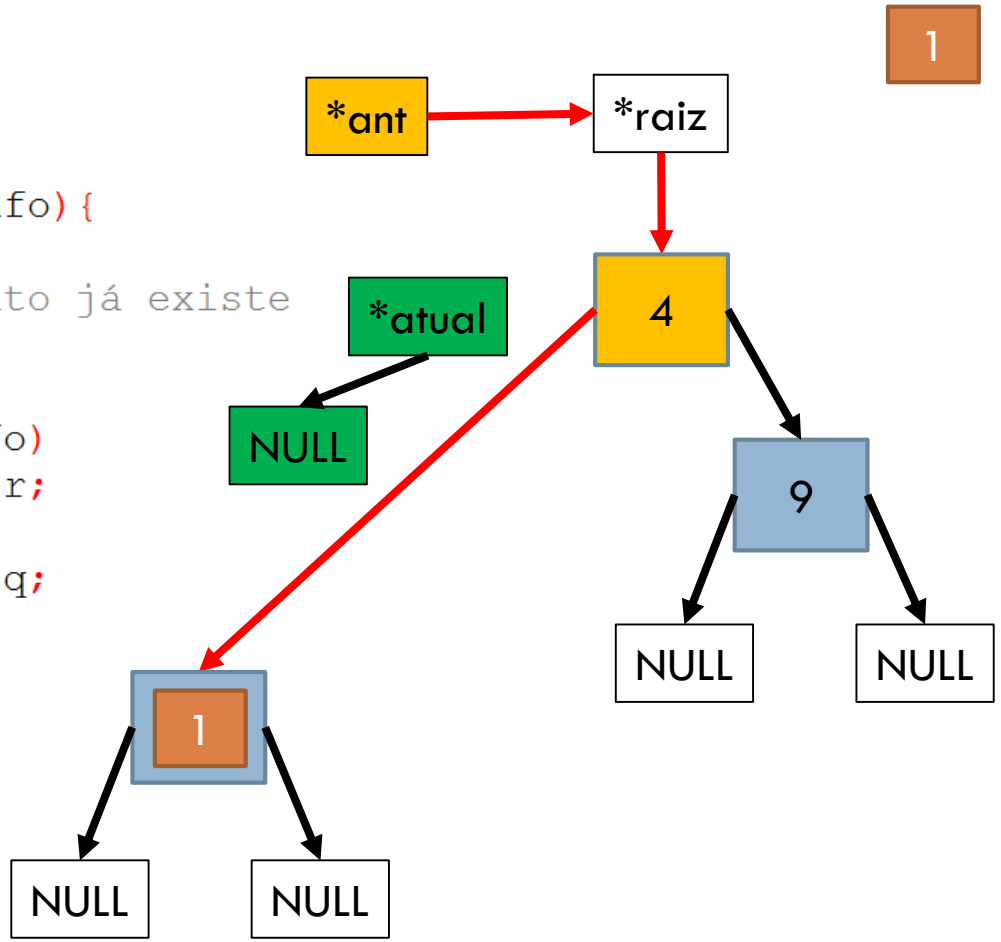

```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```





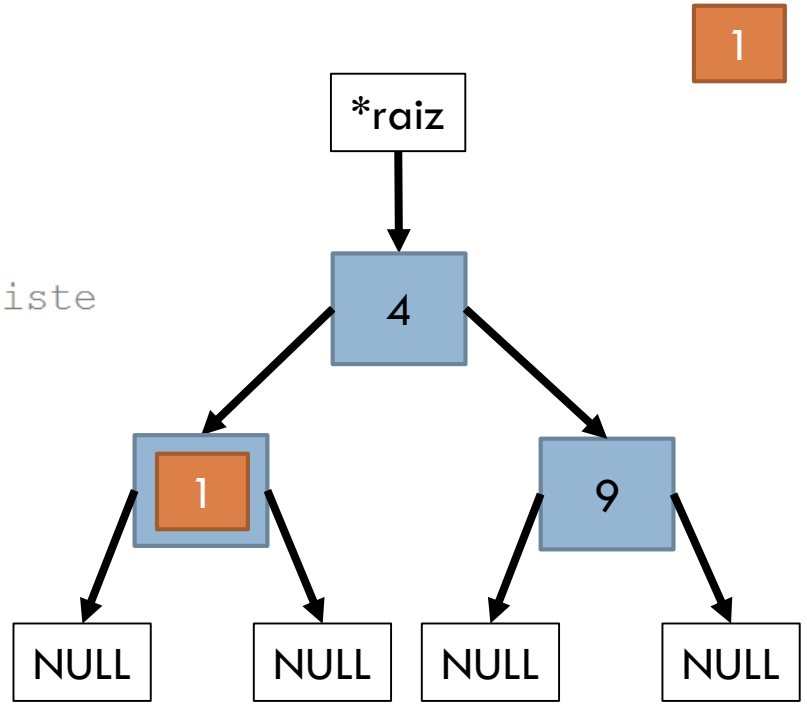
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

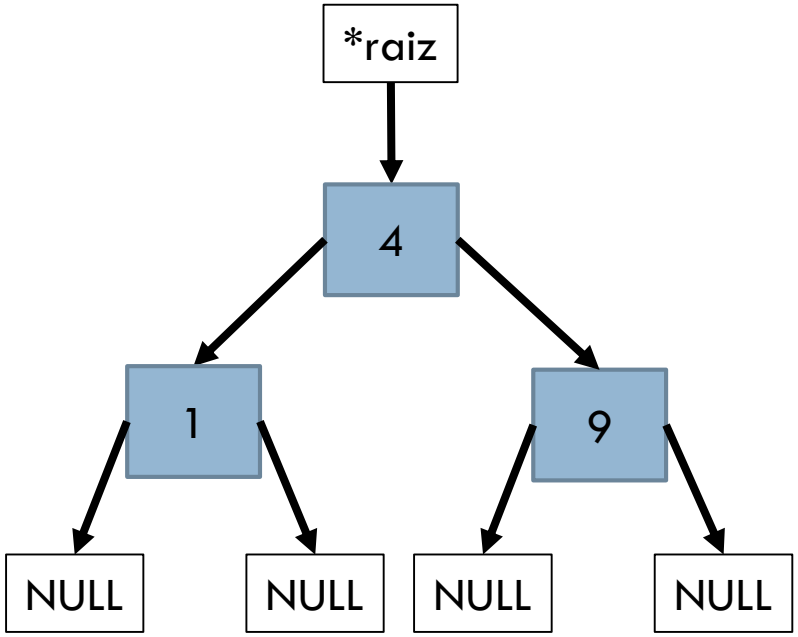


Inserção em ABB


```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

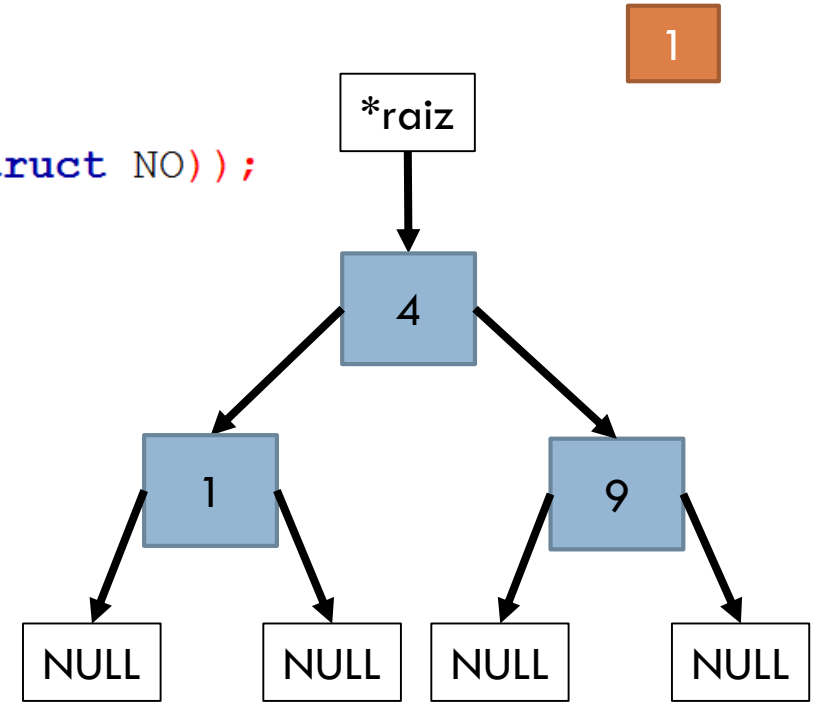


Inserção em ABB



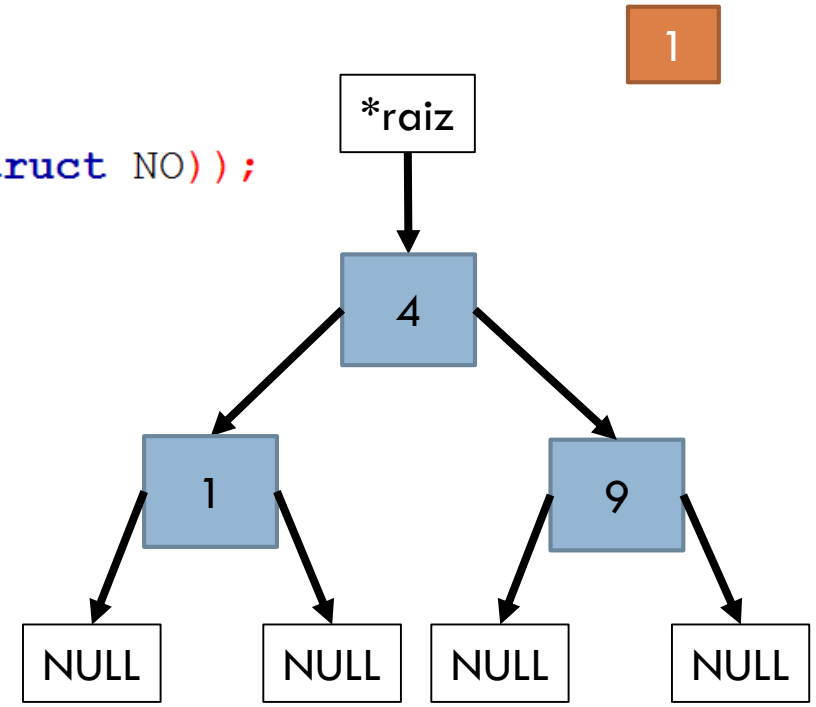
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



Inserção em ABB

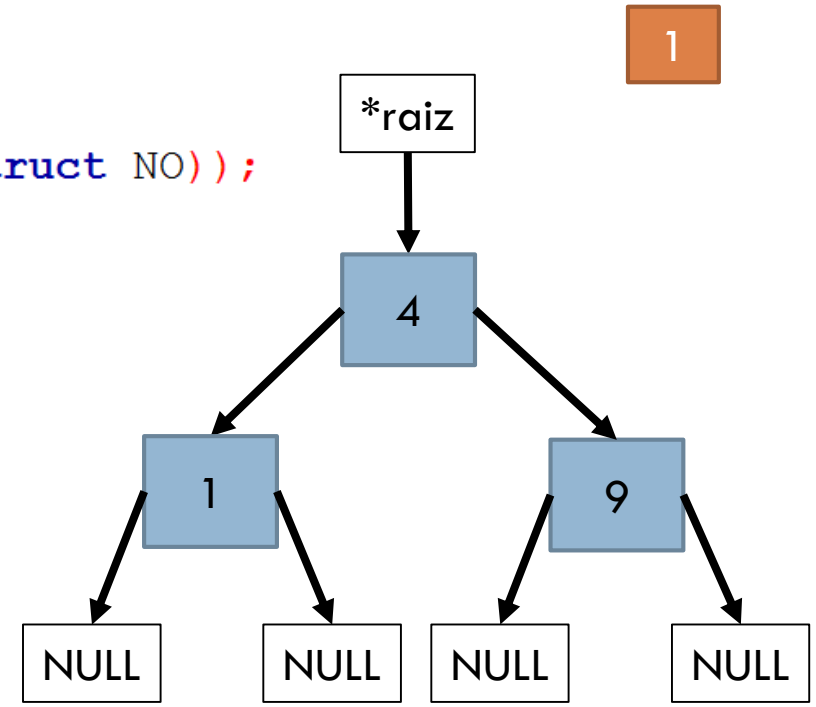

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



Inserção em ABB

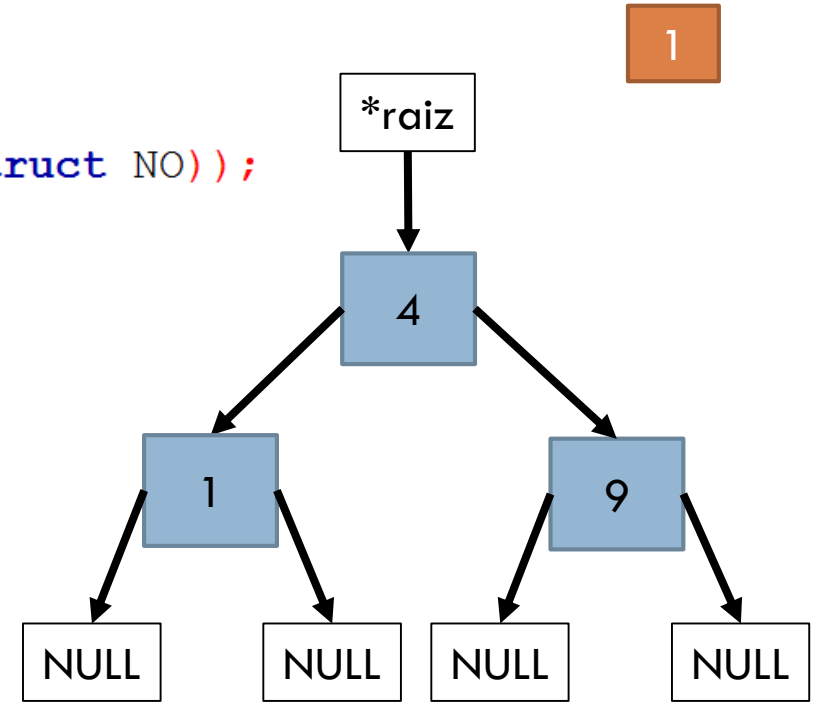

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```

novo



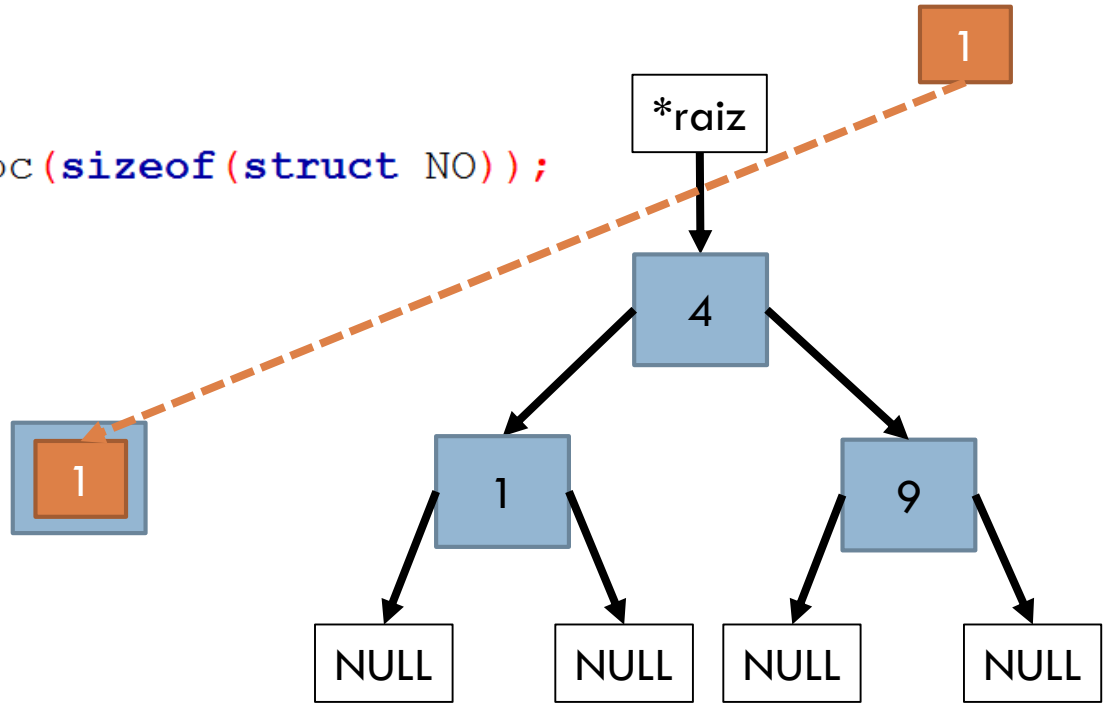
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
           [...]
```



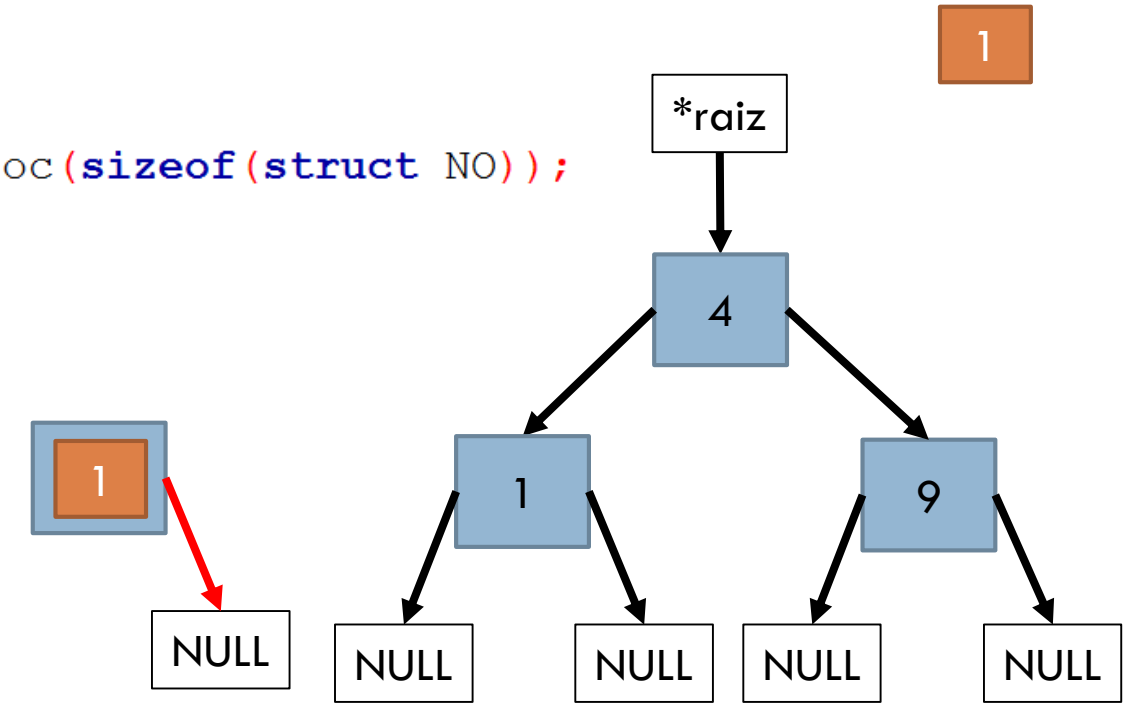
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



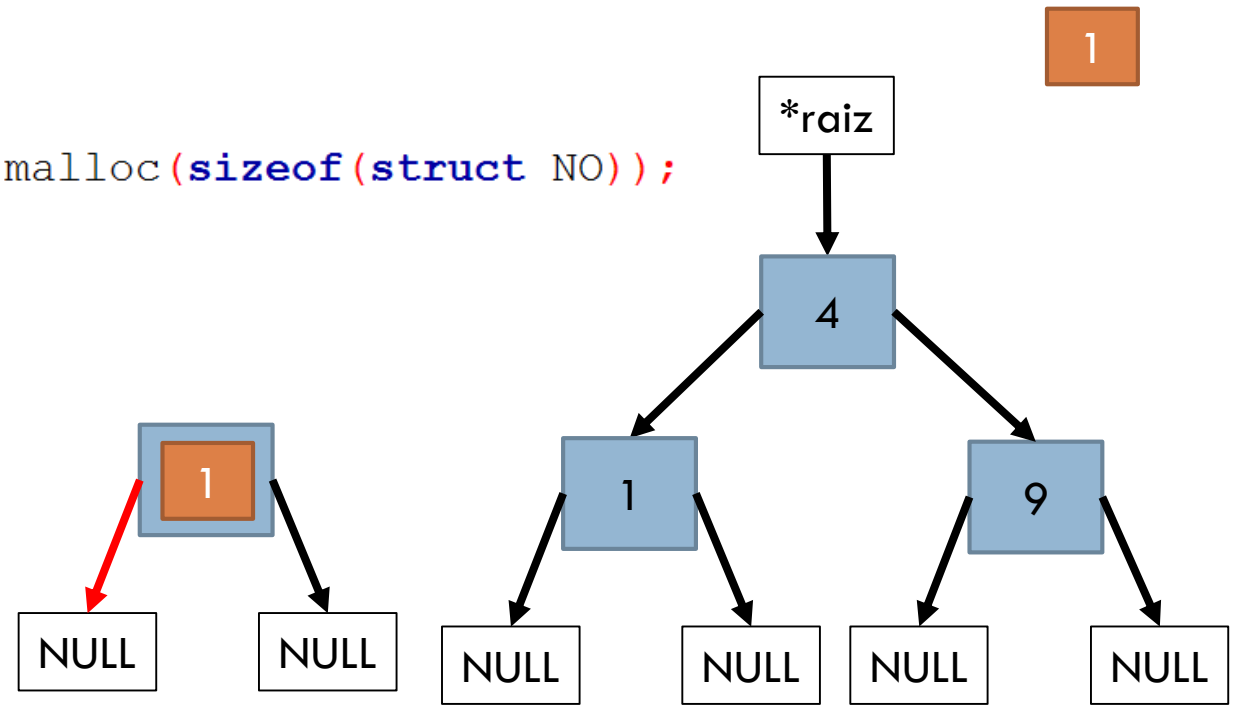
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



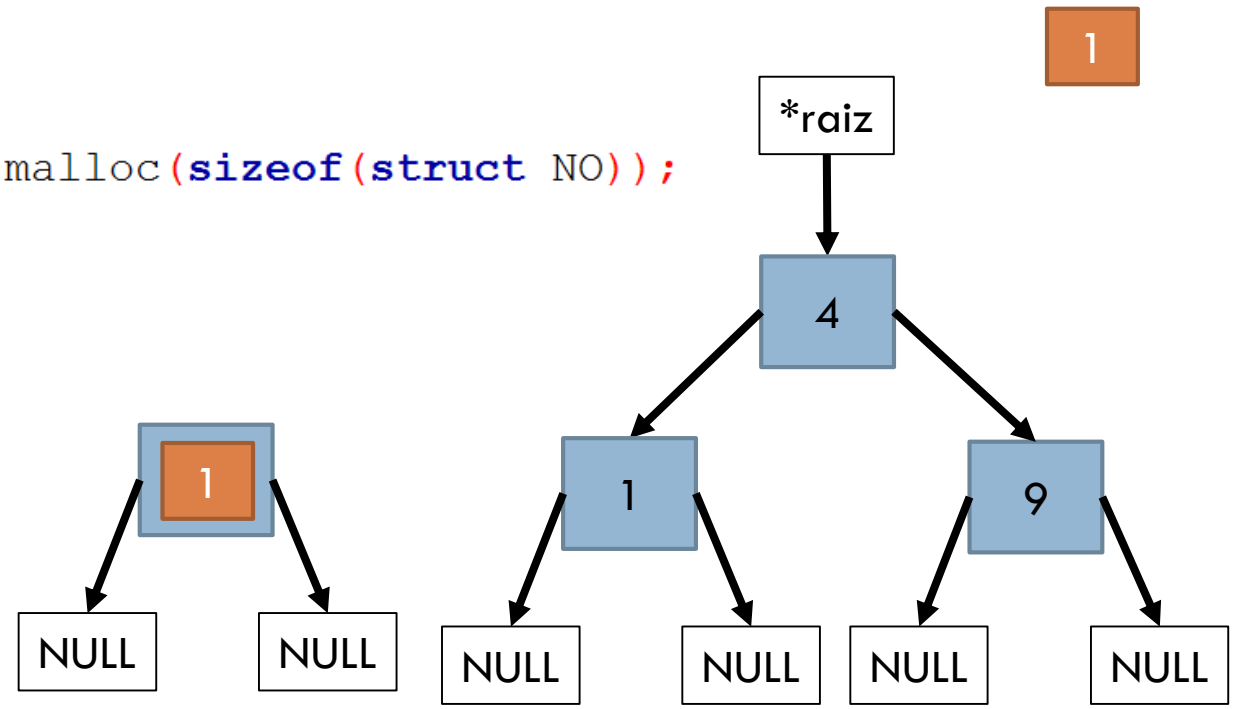
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
            [...]
```



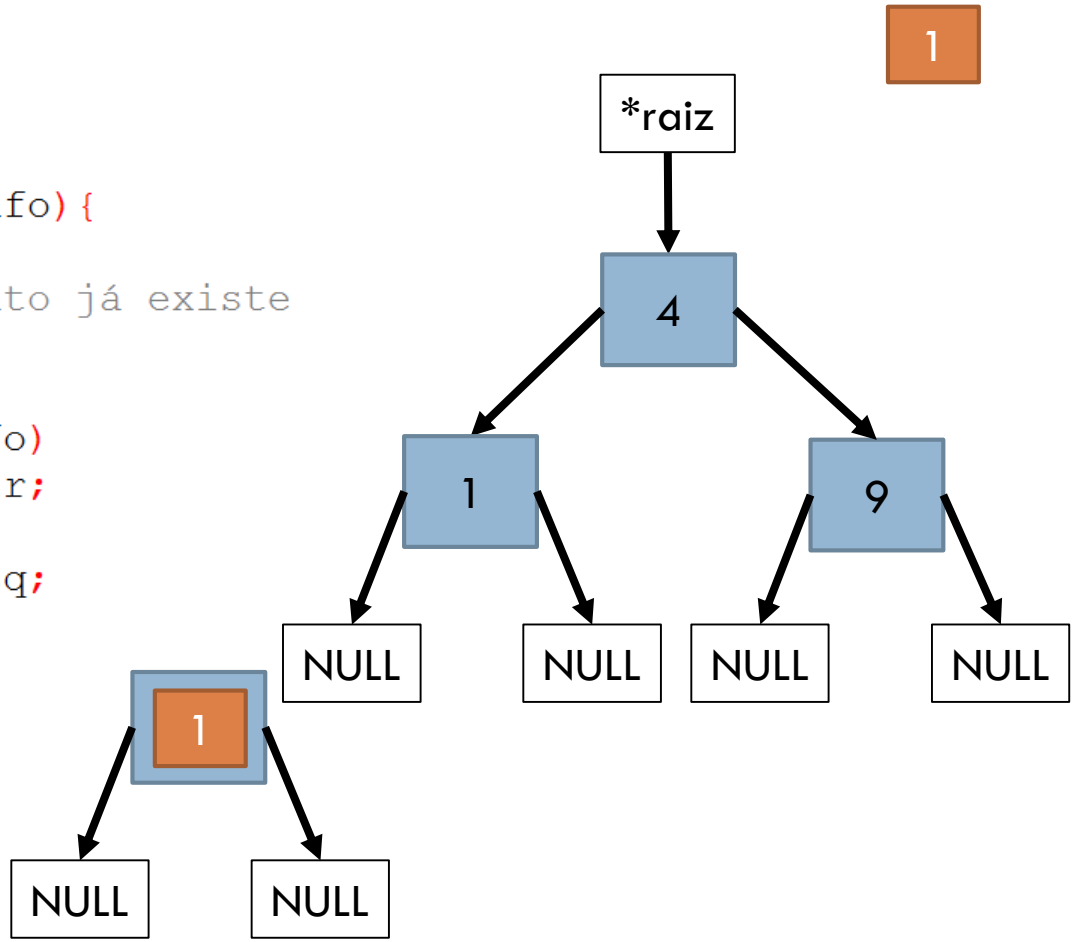
Inserção em ABB


```
int insere_ArvBin(ArvBin* raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof(struct NO));
    if(novo == NULL)
        return 0;
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if(*raiz == NULL)
        *raiz = novo;
          [...]
```



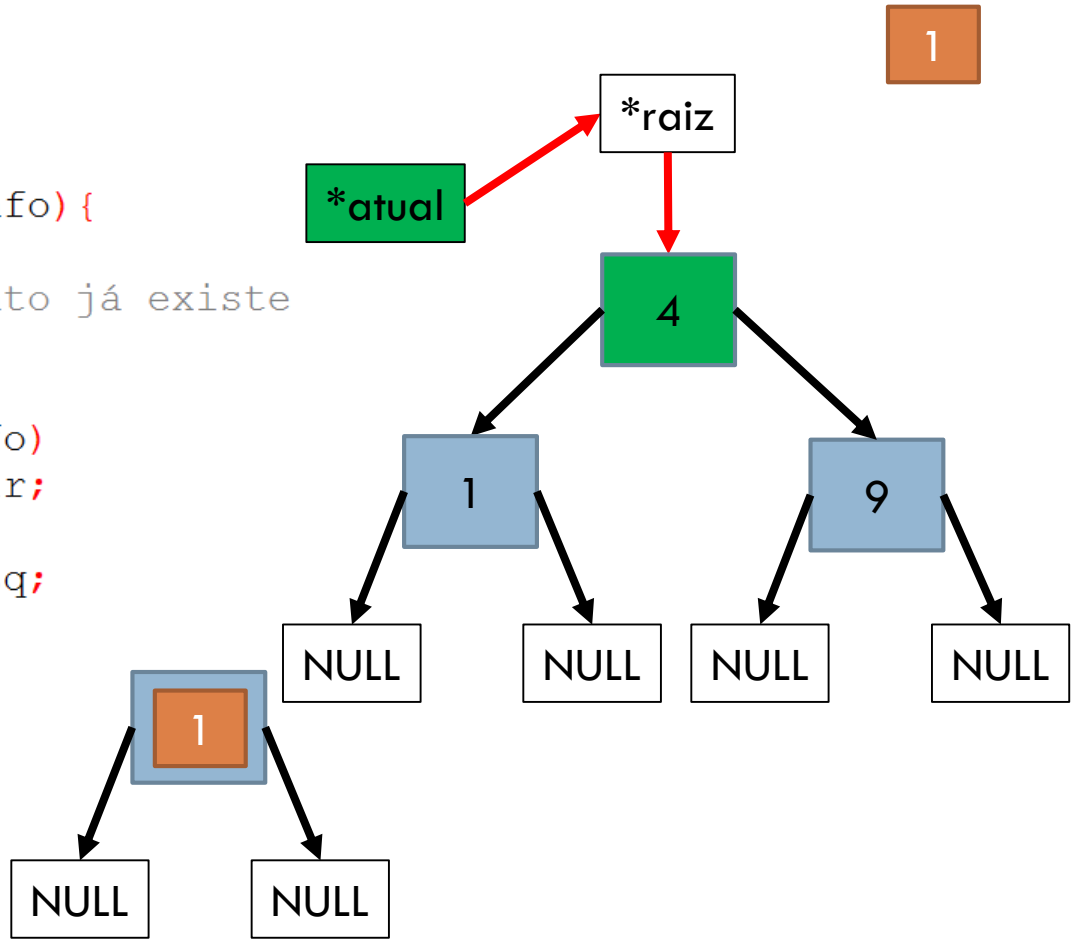
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



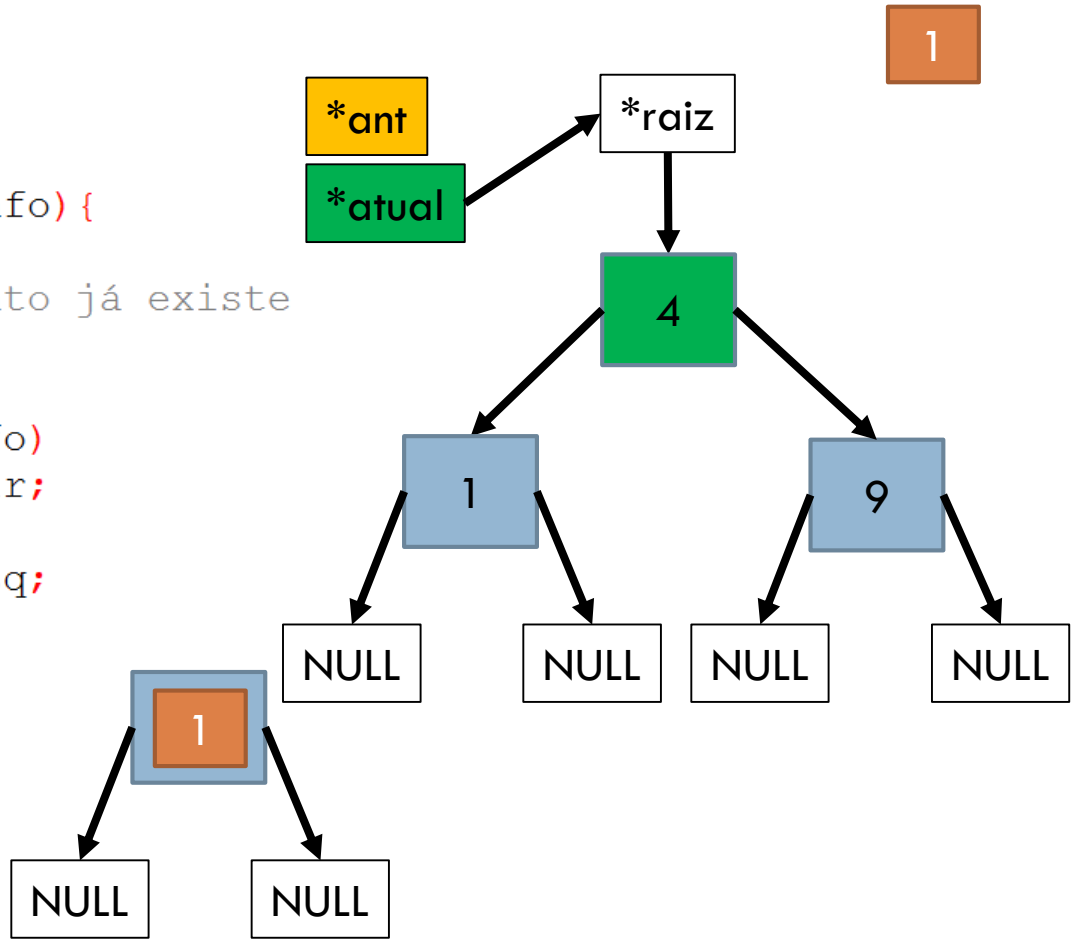
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



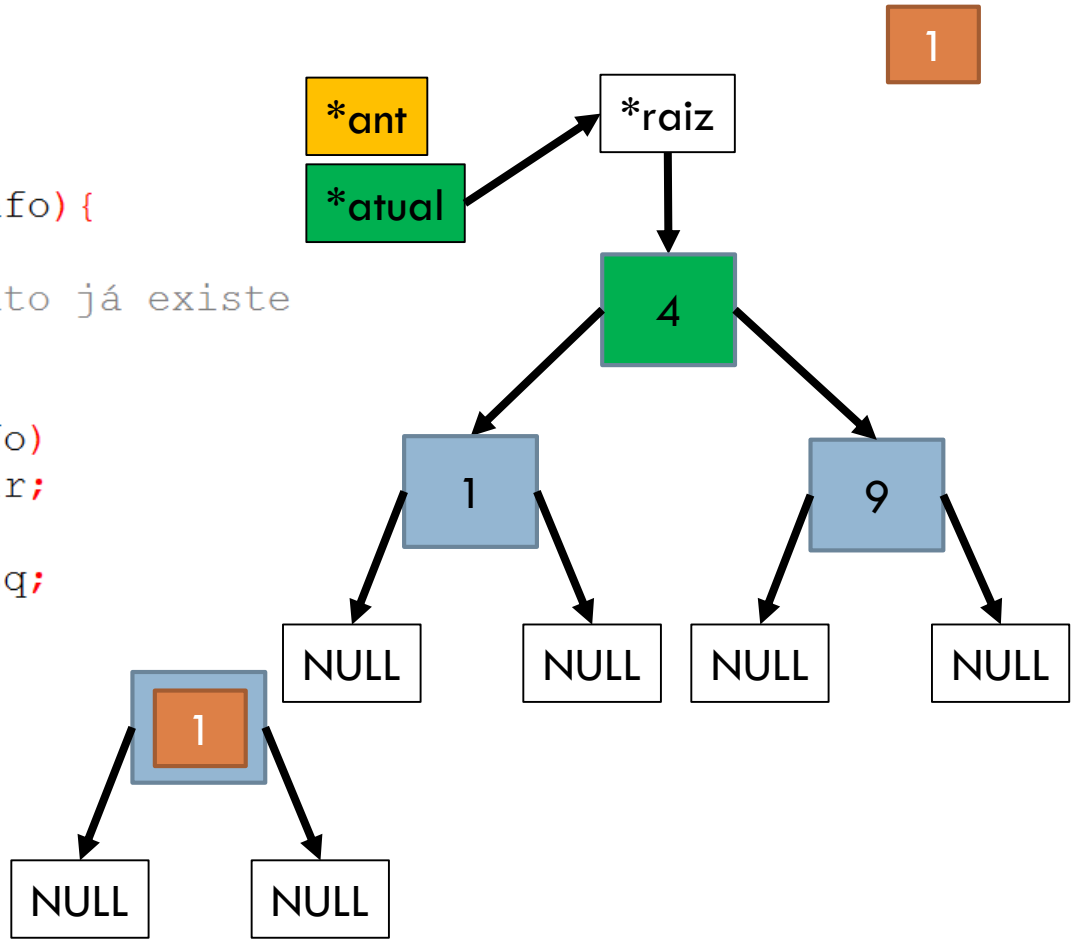
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



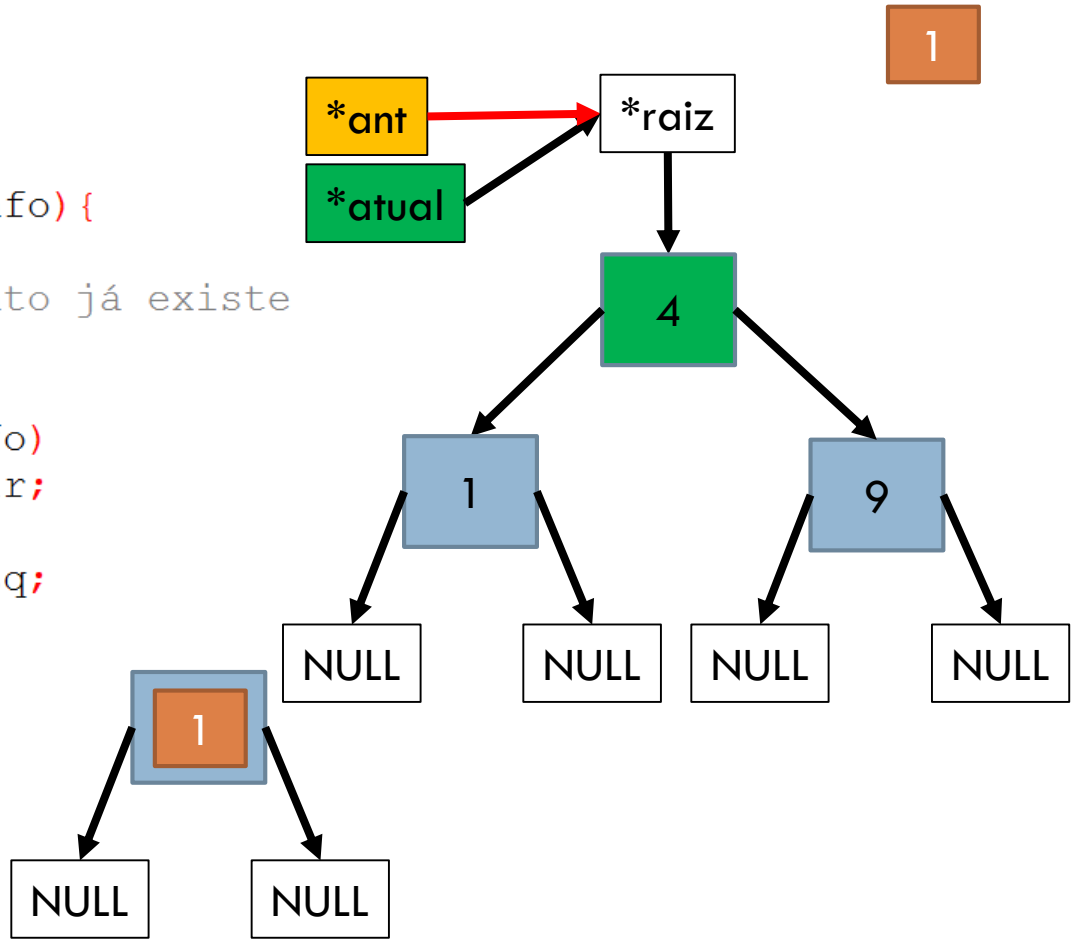
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



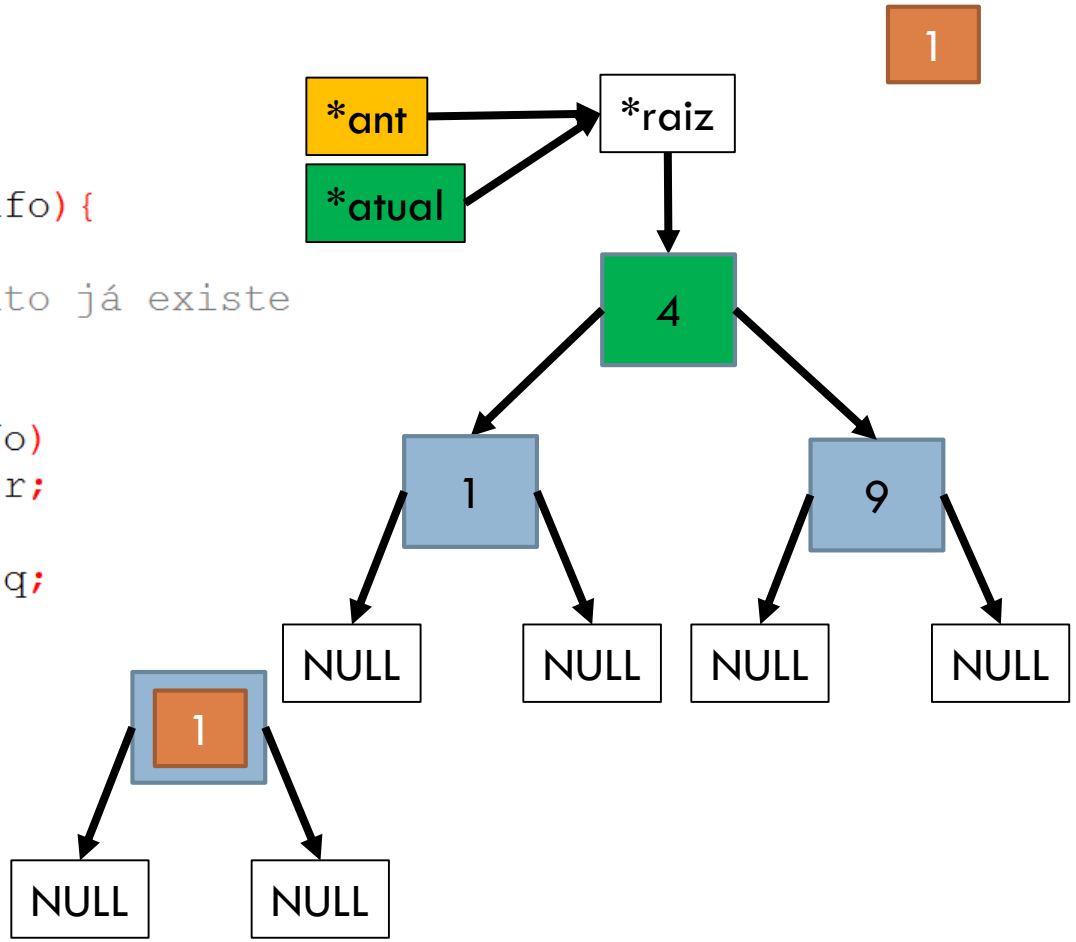
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



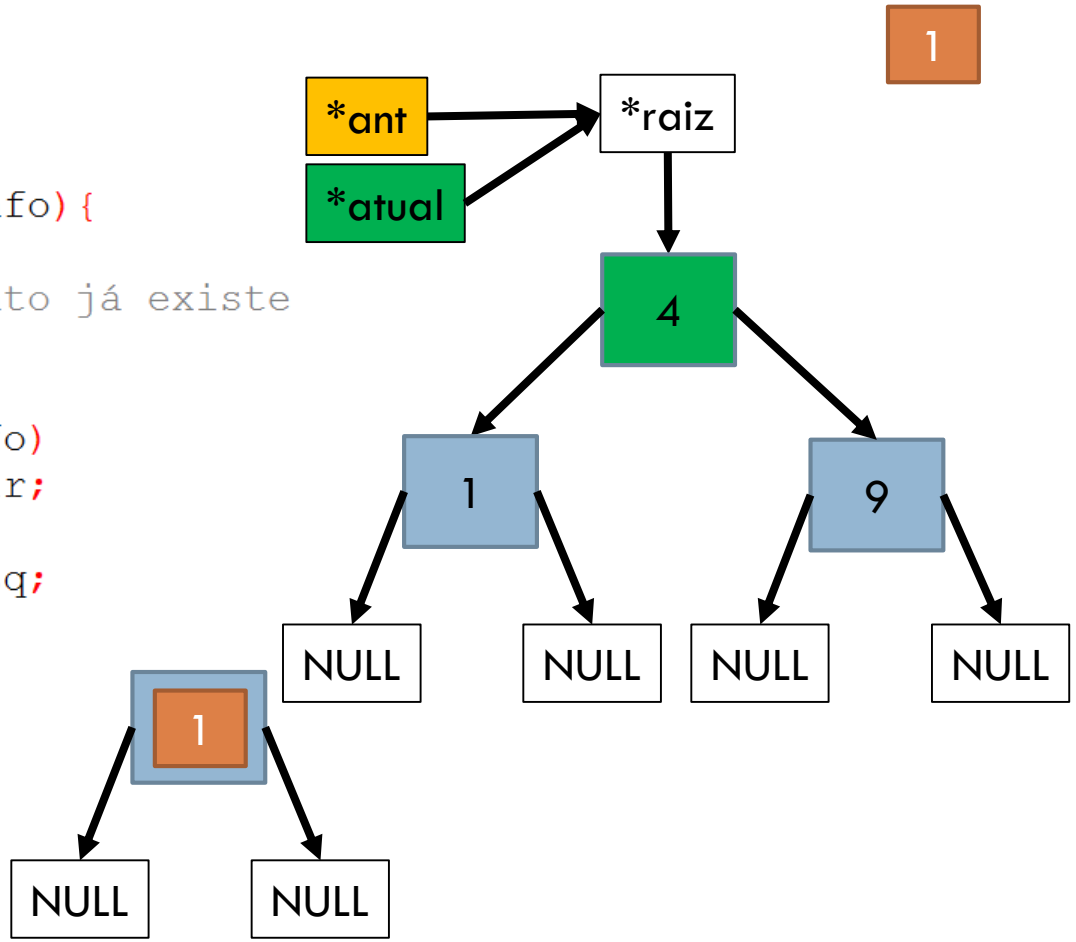
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



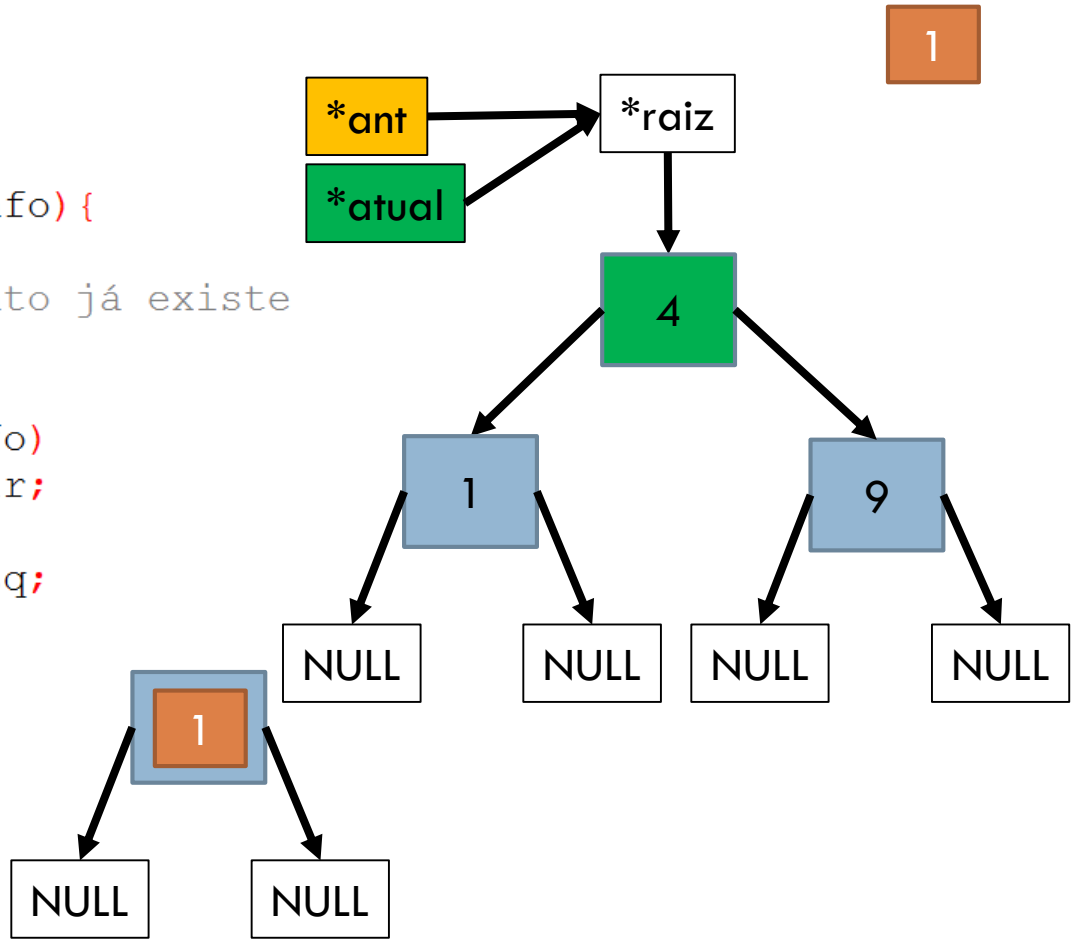
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



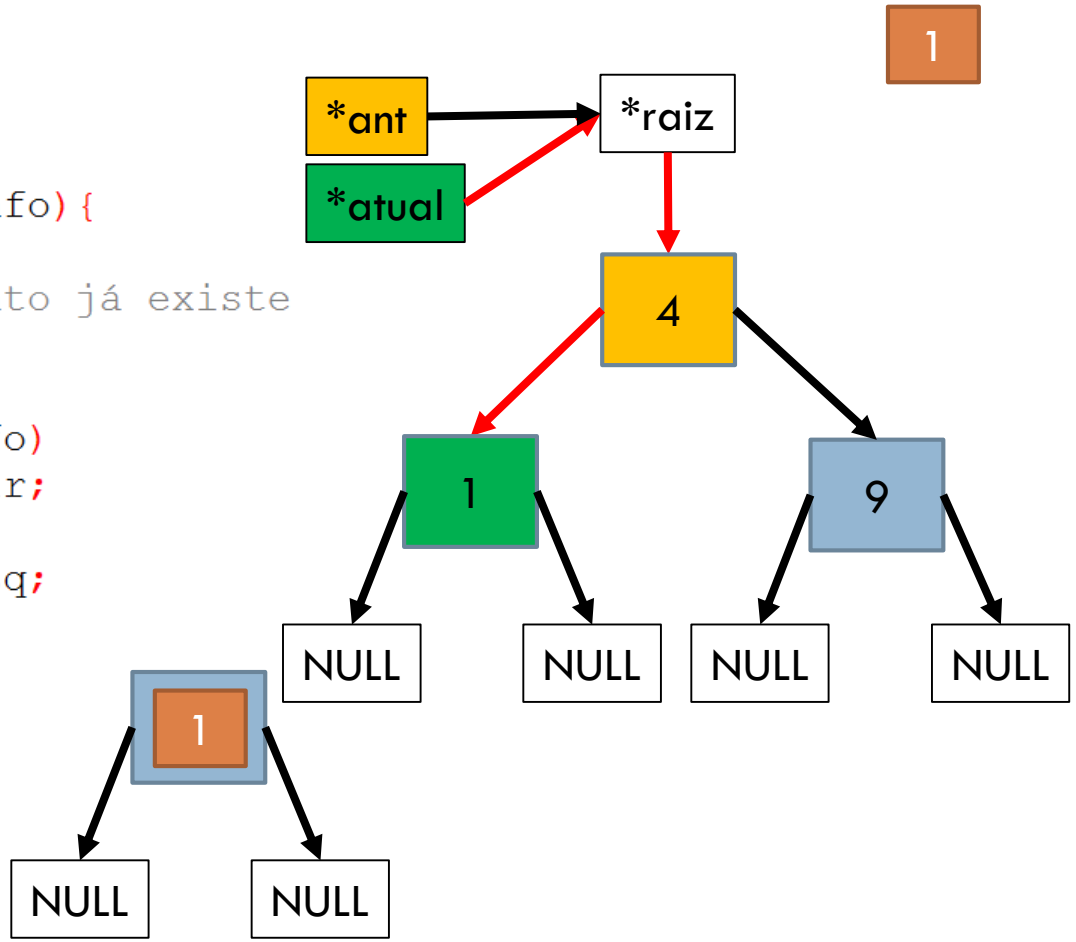
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



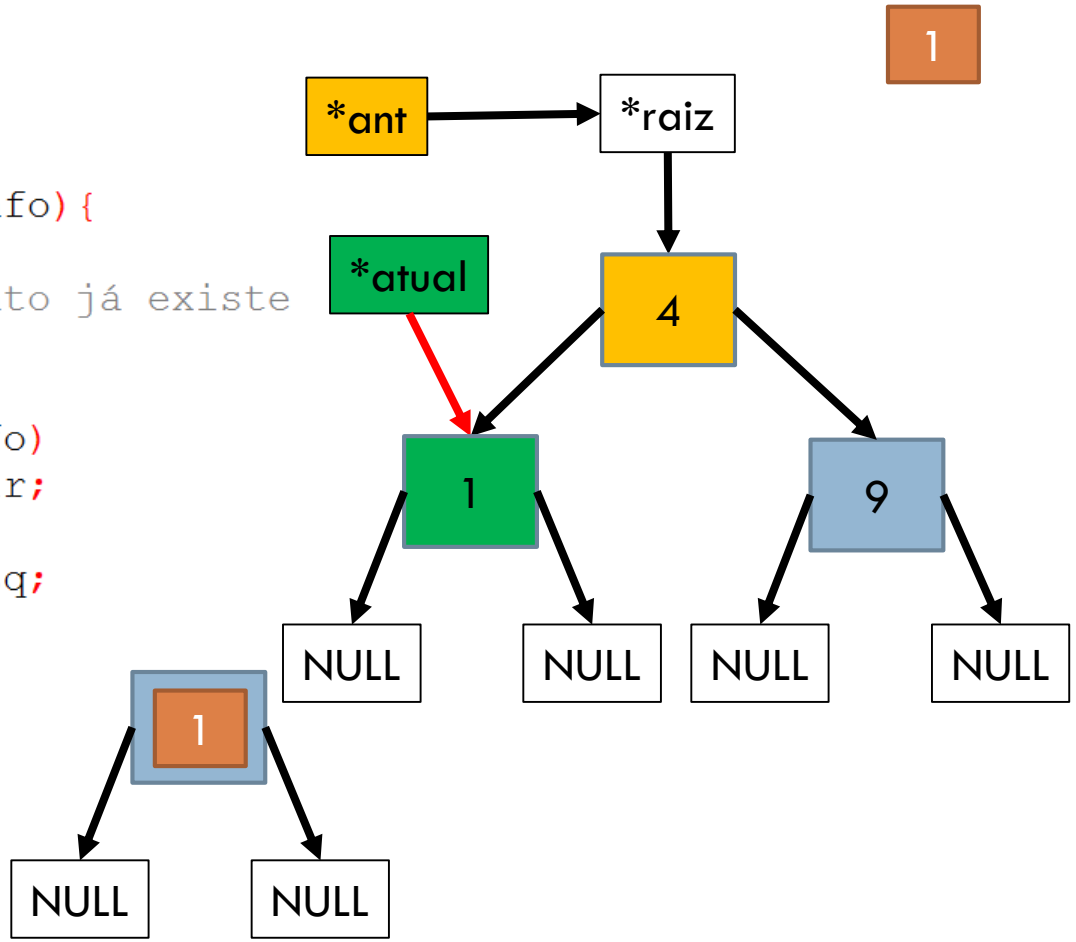
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



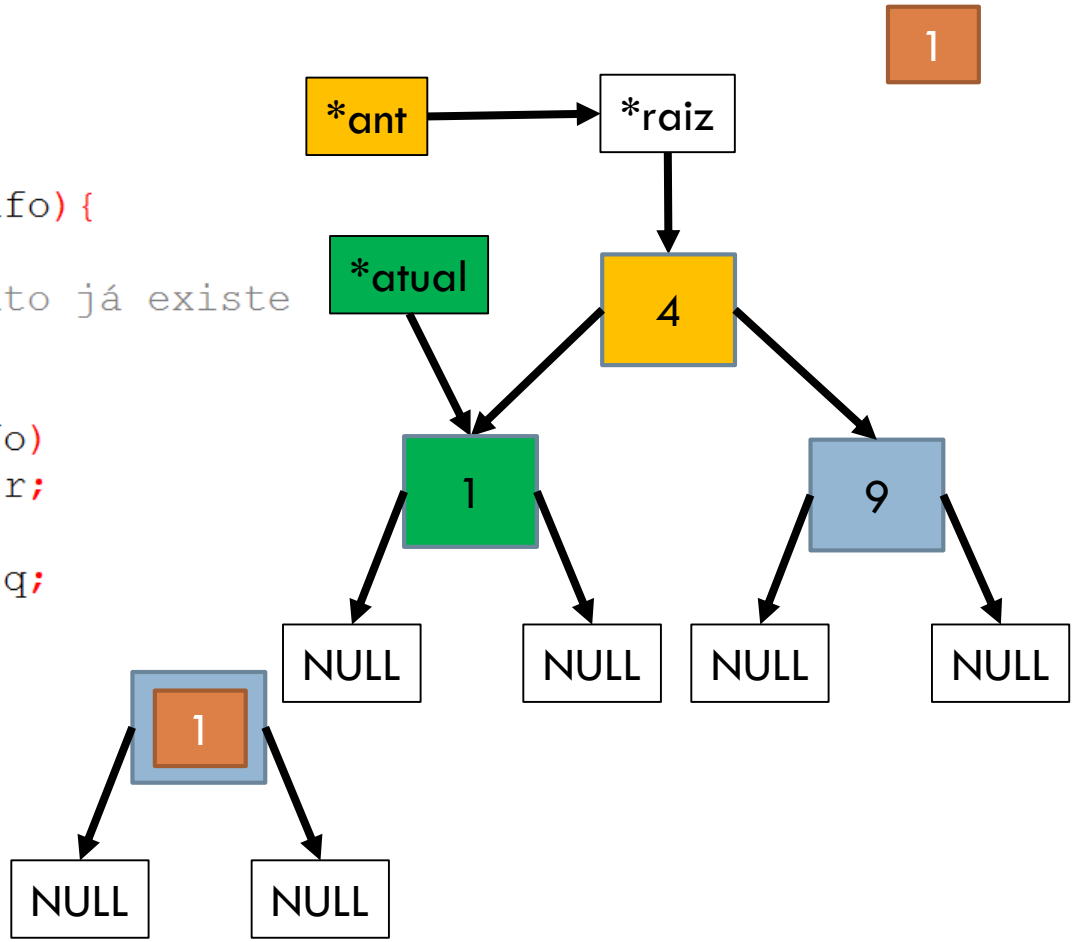
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



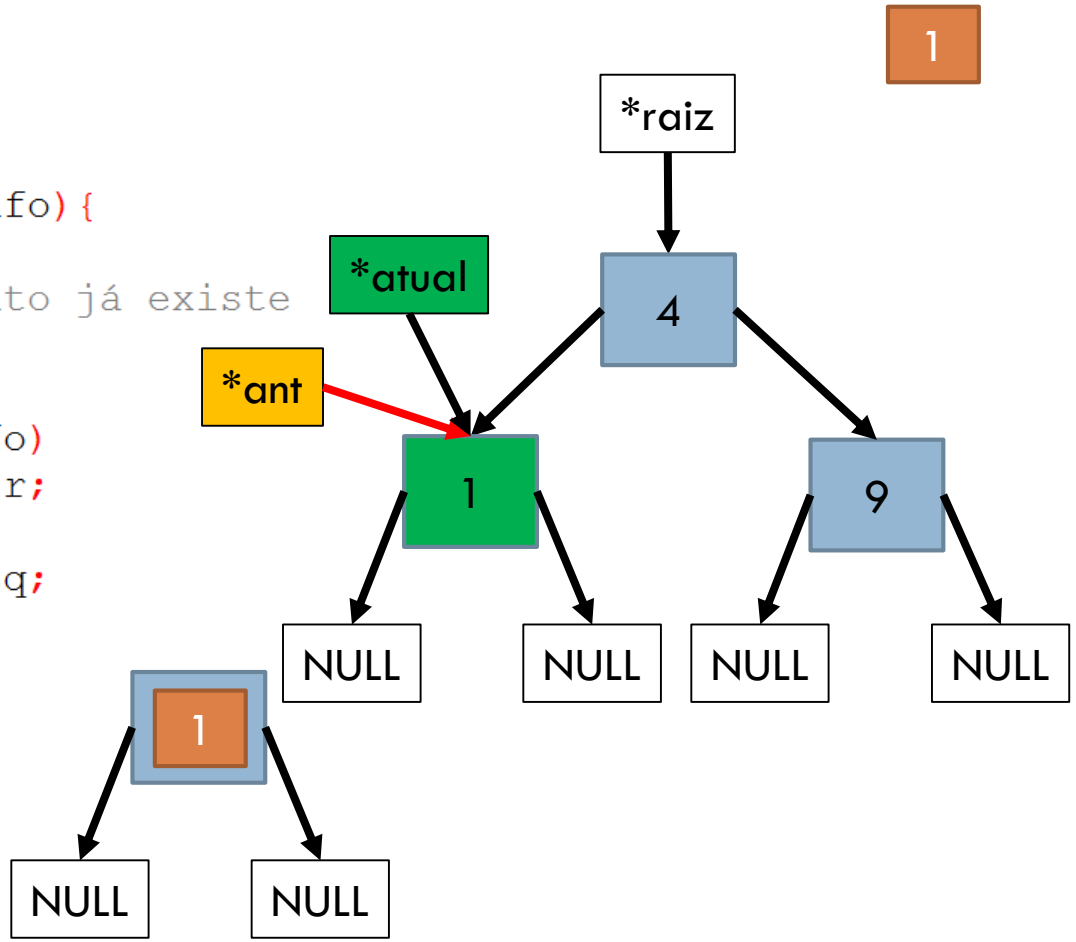
Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



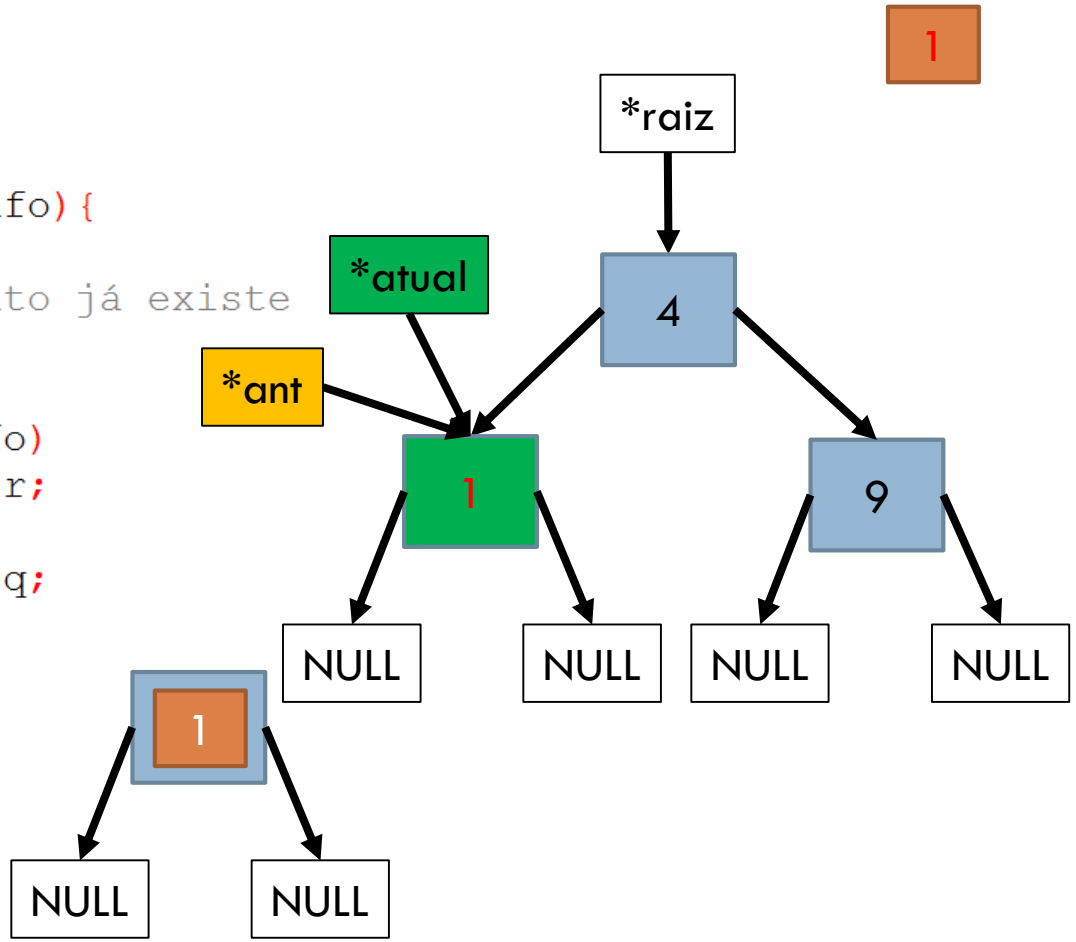
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



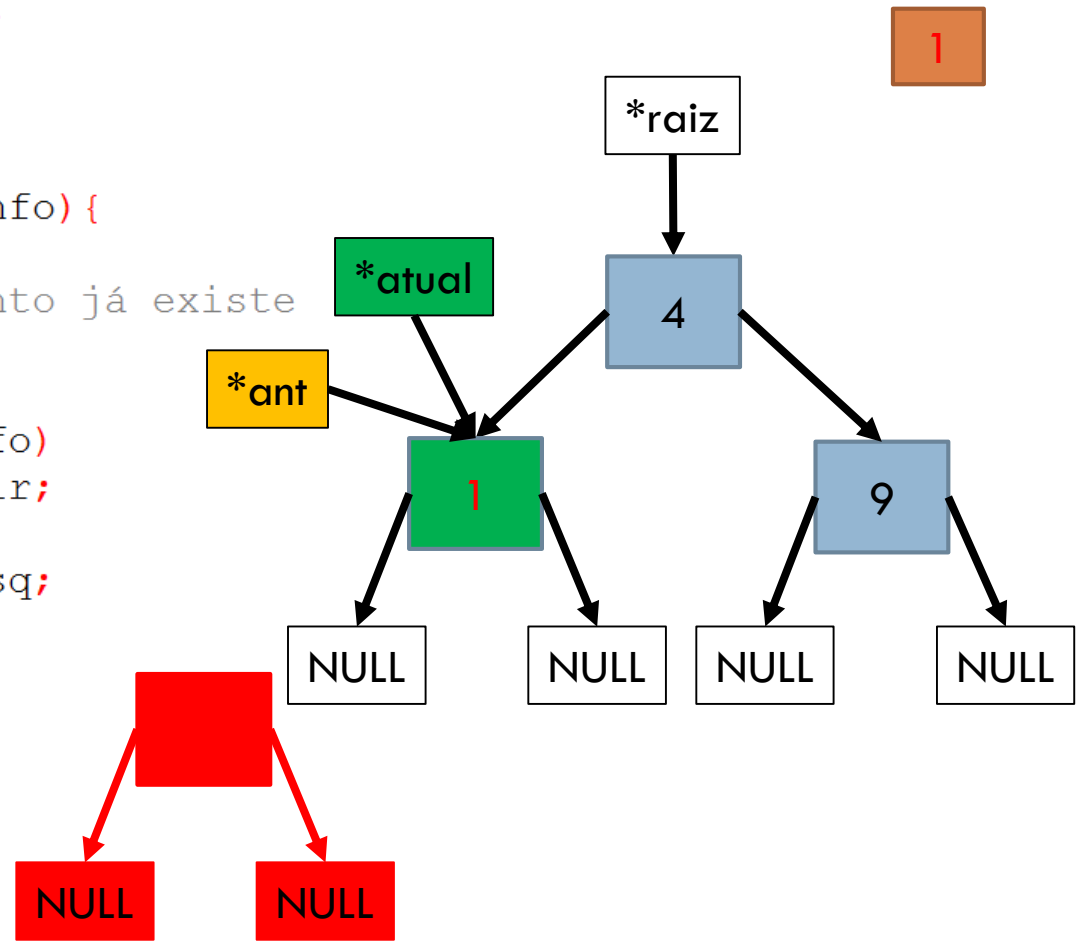
Inserção em ABB


```
[...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



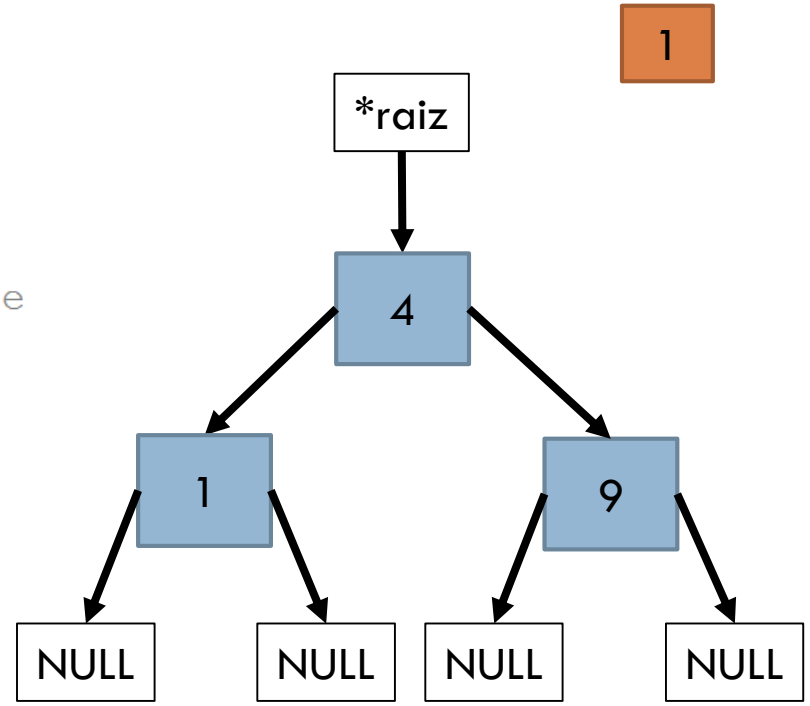
Inserção em ABB


```
        [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```

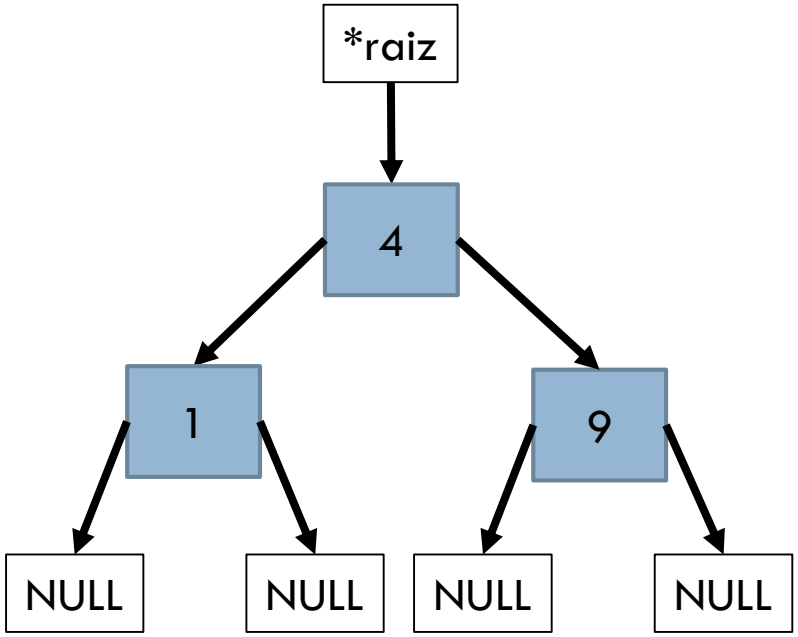


Inserção em ABB


```
            [...]
else{
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0;//elemento já existe
        }

        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB



Busca em ABB

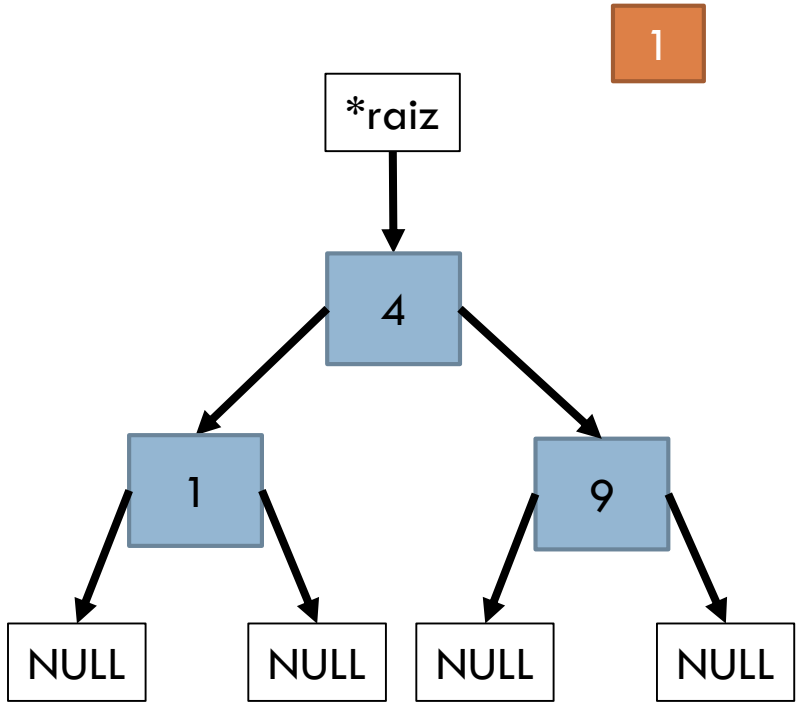
- Receber a raiz de uma árvore e uma chave.
- Percorrer a árvore até encontrar o nó com a chave especificada.
- Retornar o endereço do nó ou NULL.

Busca em ABB

- Algoritmo busca(*raiz, valor):
  - Se a posição está vazia (NULL), dado não foi encontrado e retorna NULL;
  - Se a chave é igual ao valor passado, retorna ponteiro para o nó;
  - Se é menor que raiz, busca((*raiz)->**esquerda**, **valor**);
  - Se é maior que raiz, busca((*raiz)->**direita**, **valor**);

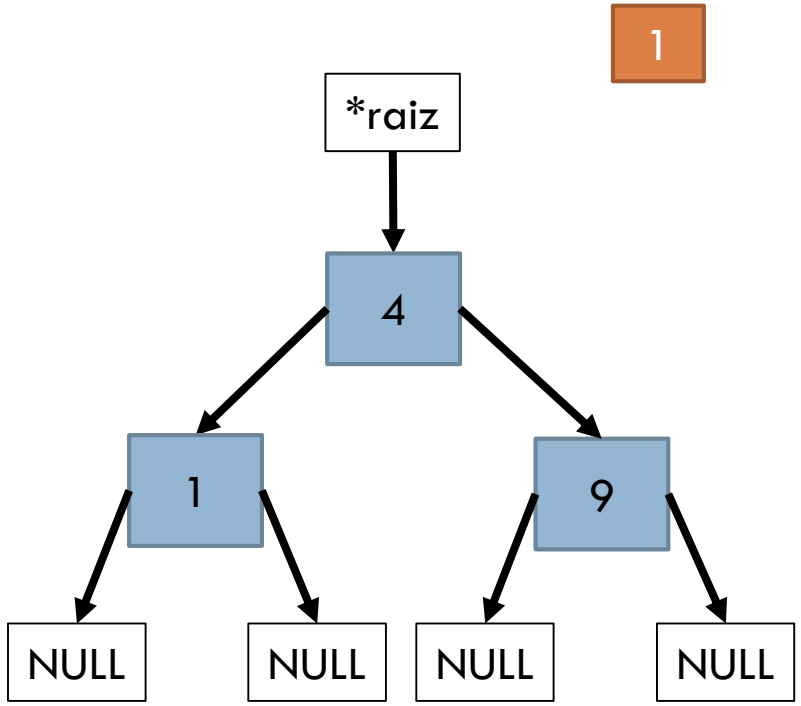
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



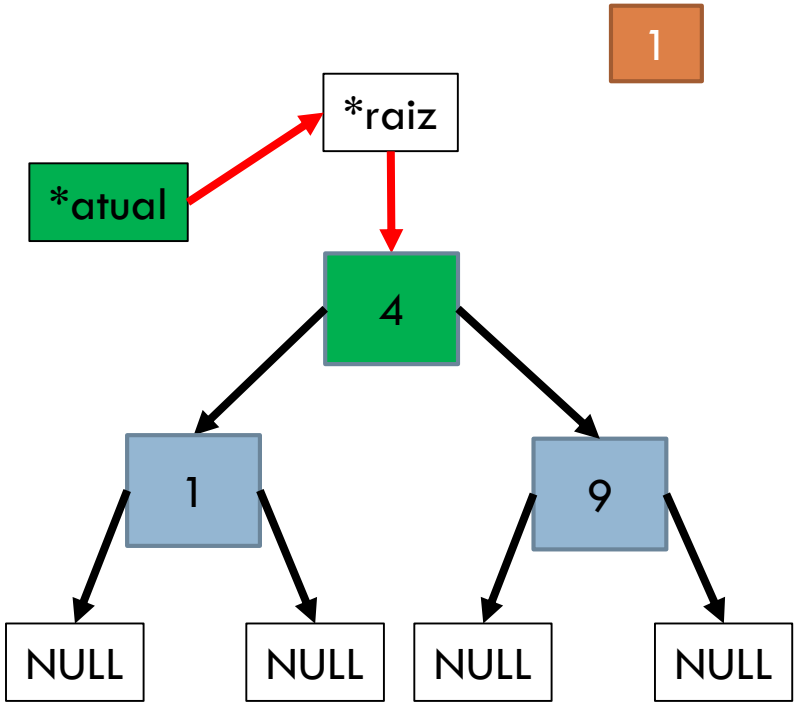
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



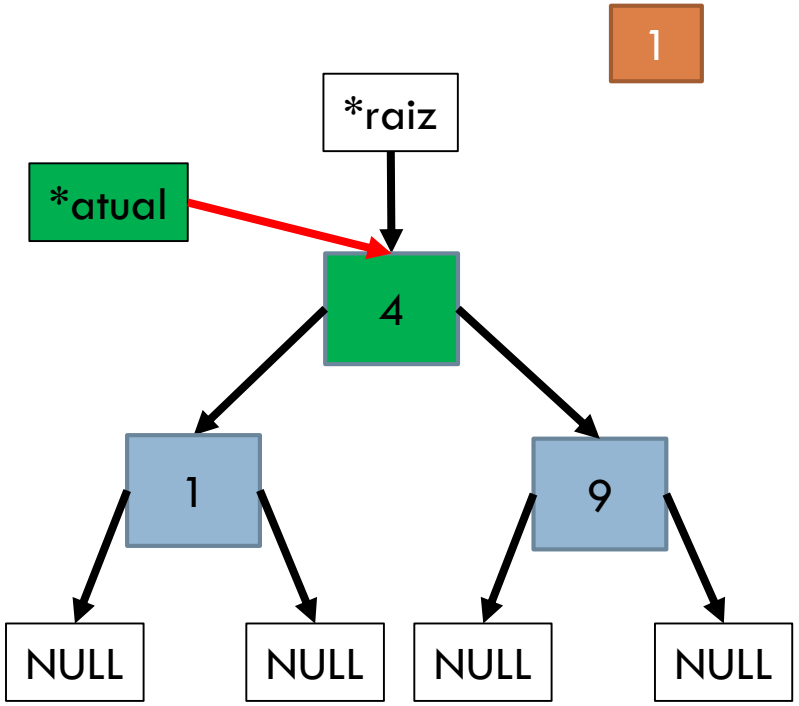
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



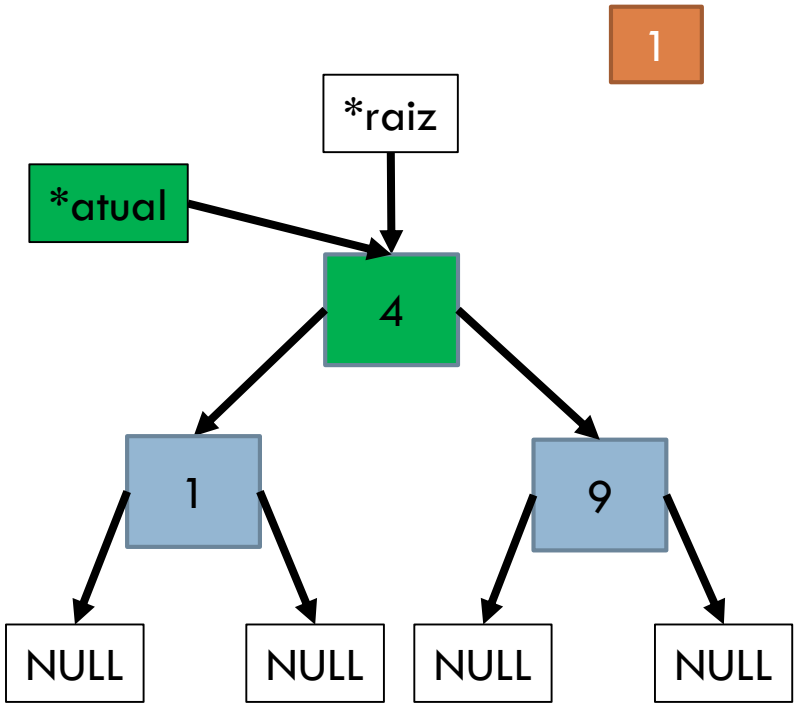
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



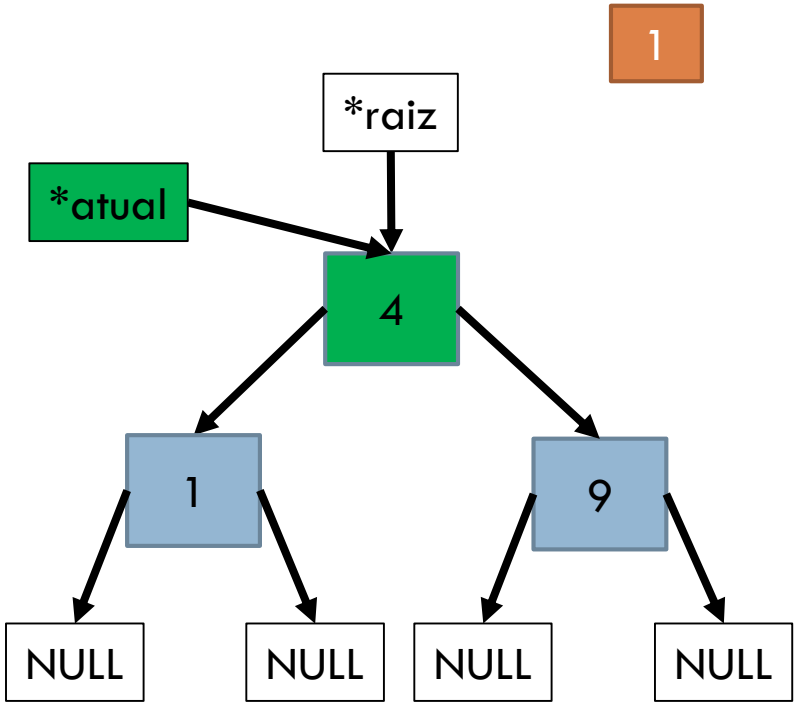
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



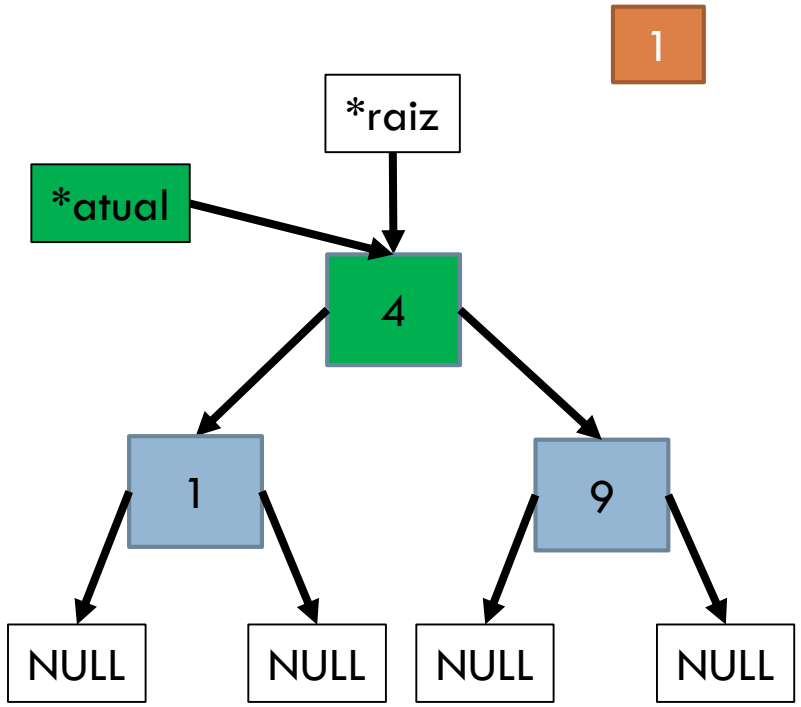
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



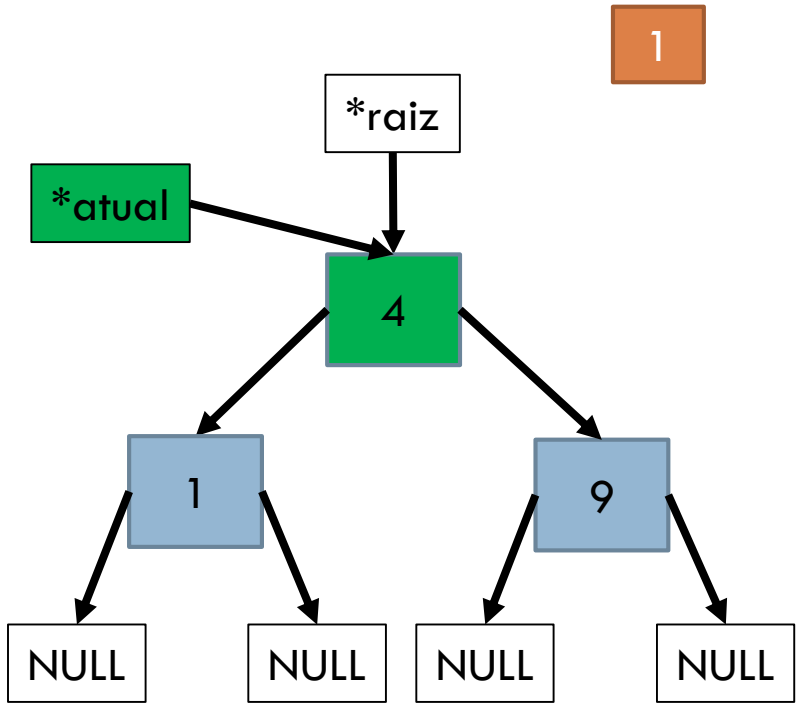
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



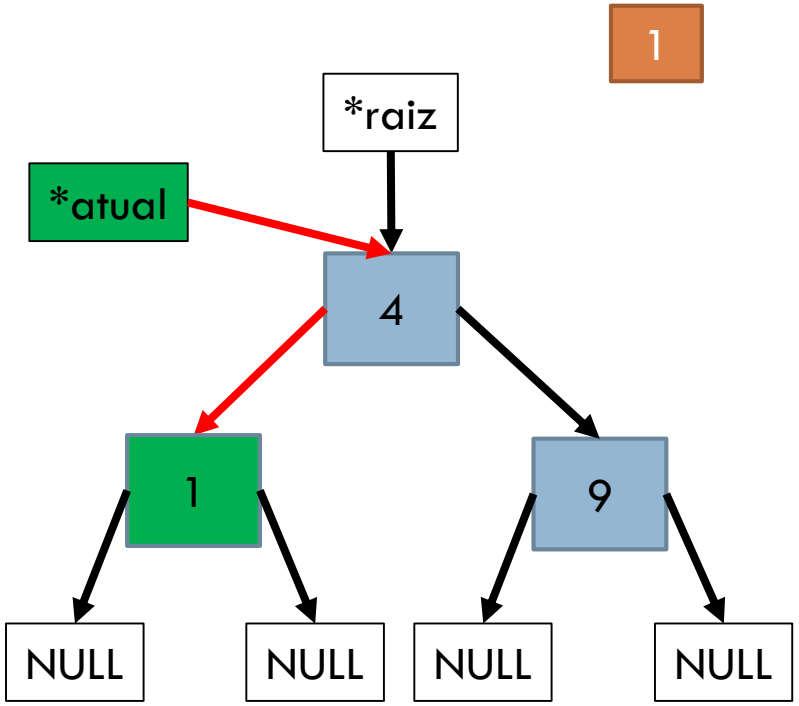
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



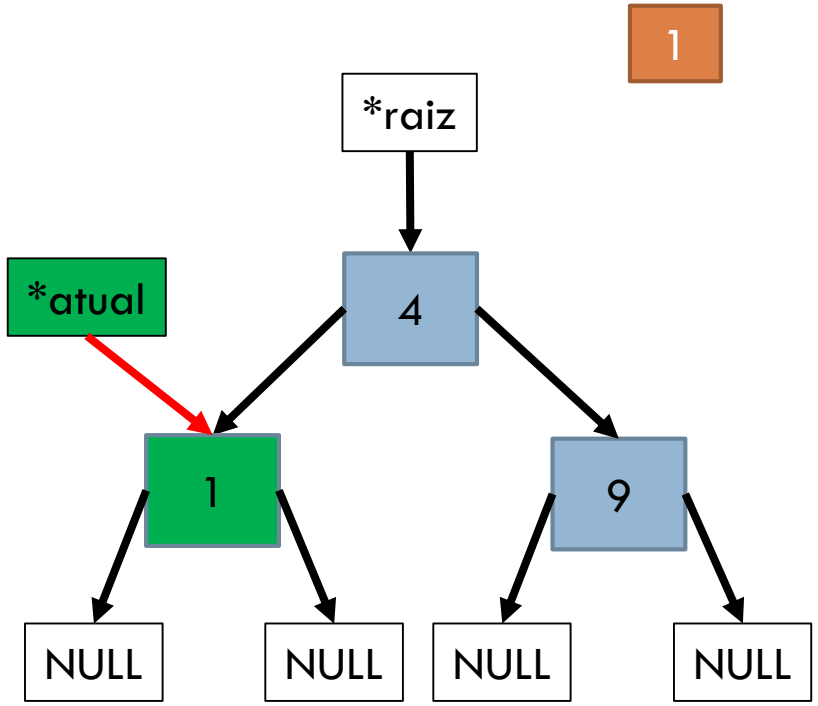
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



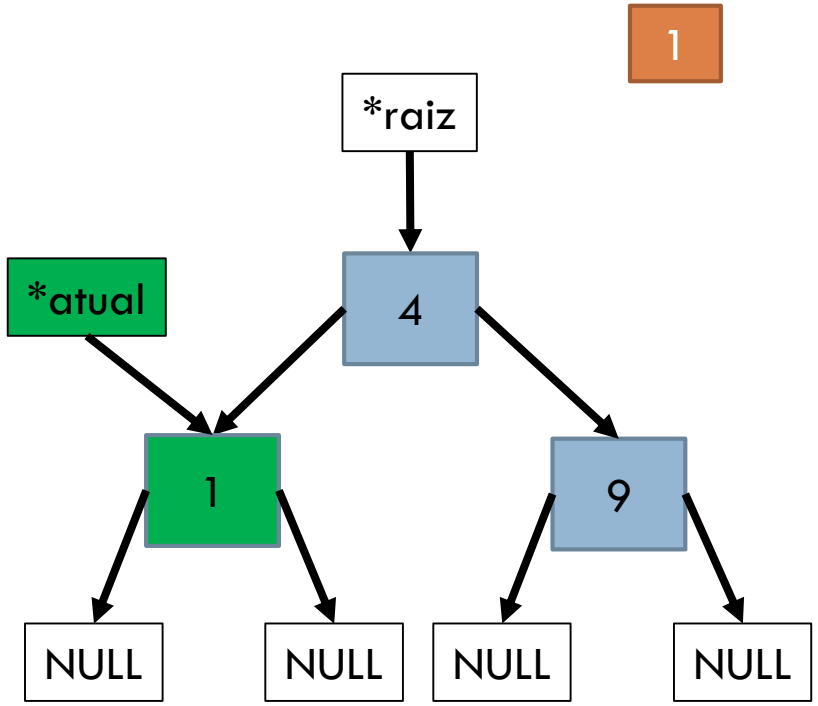
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



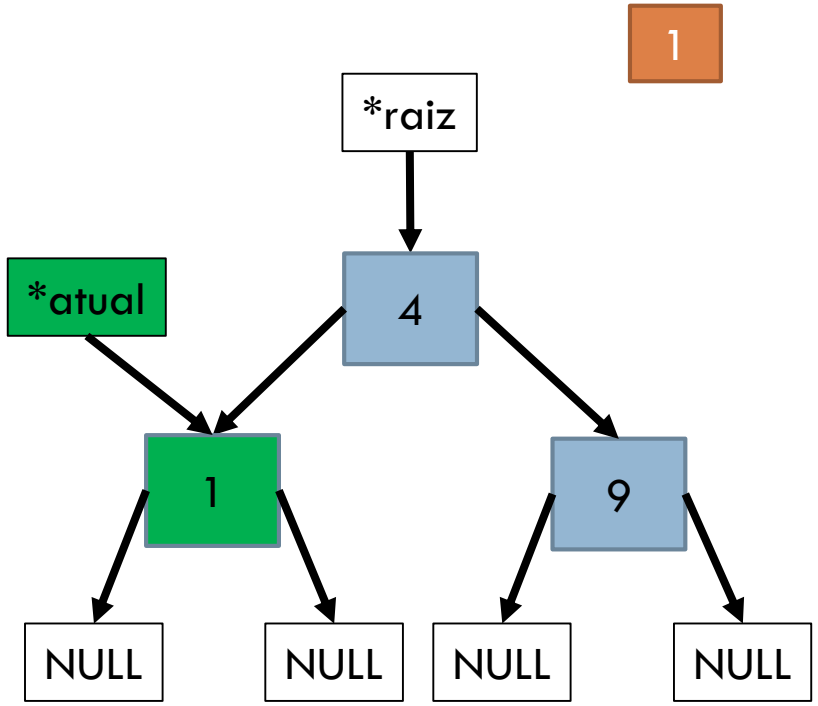
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



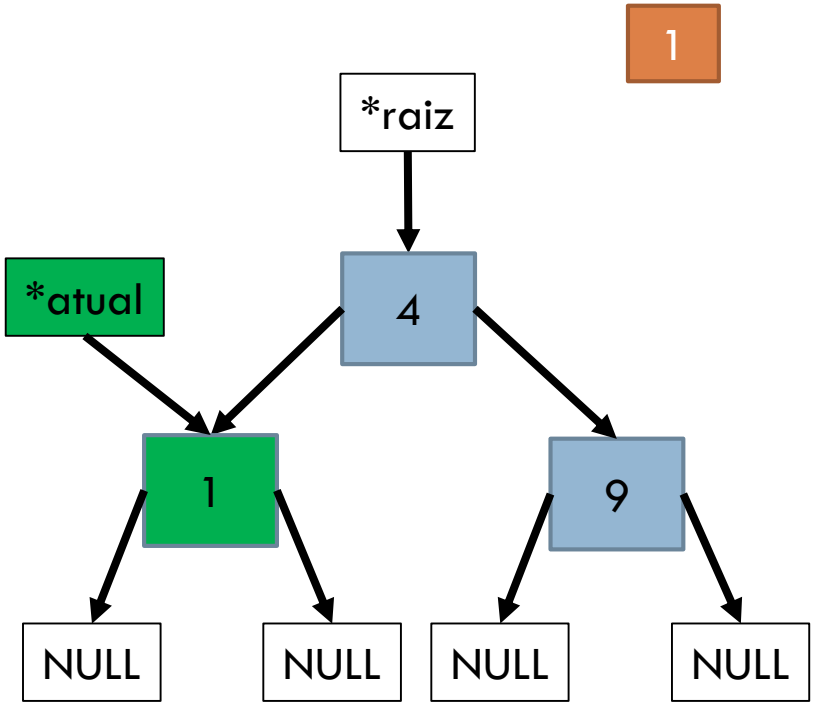
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



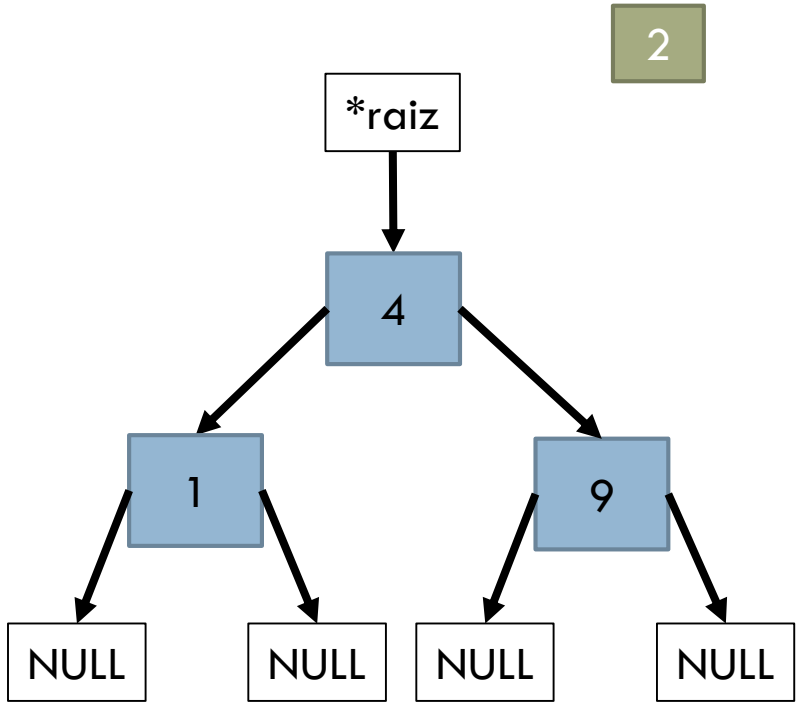
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



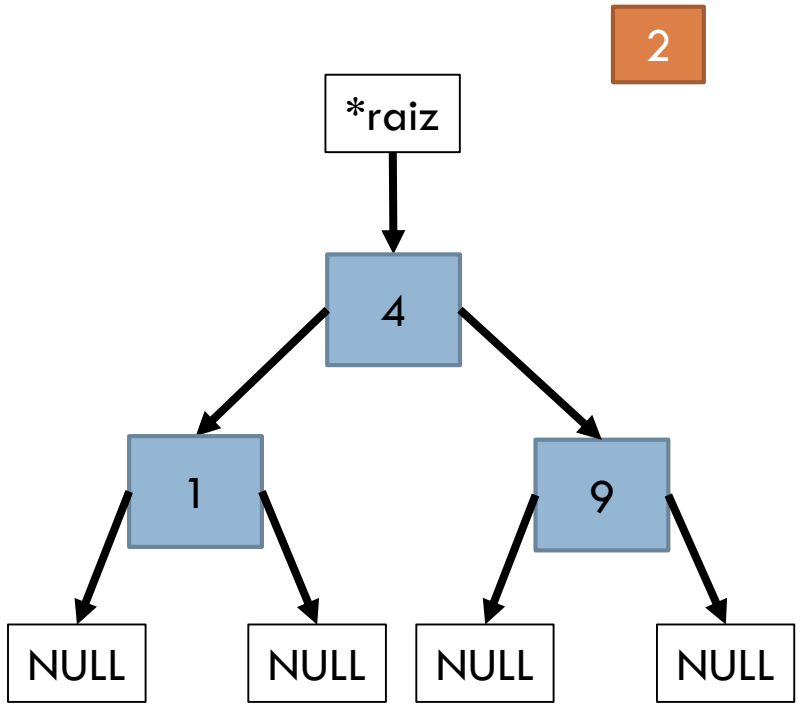
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



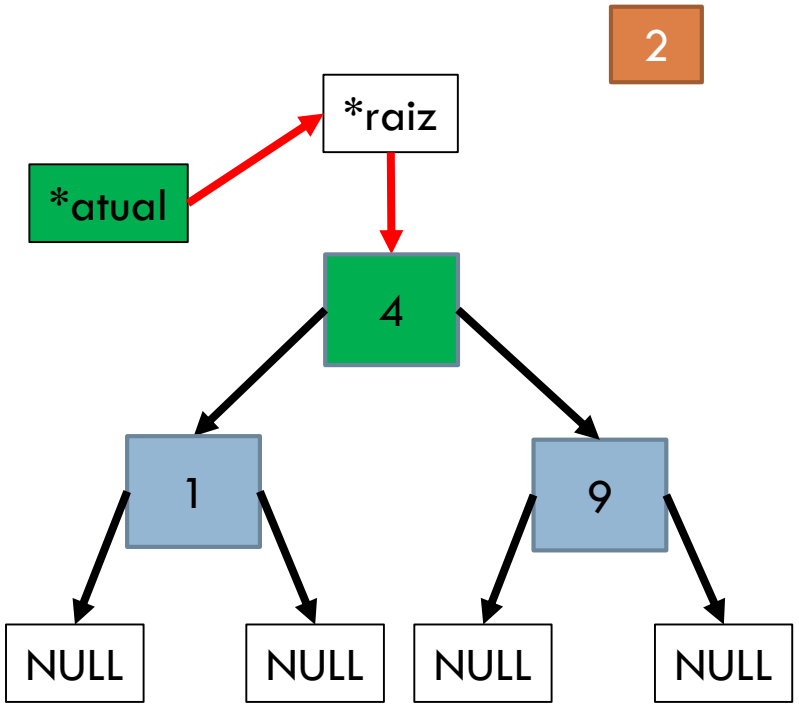
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



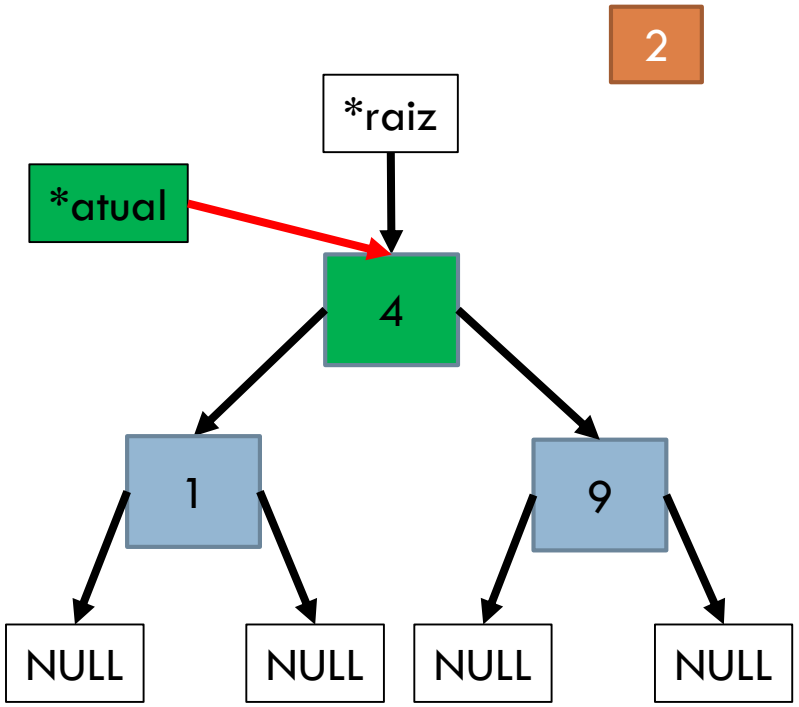
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



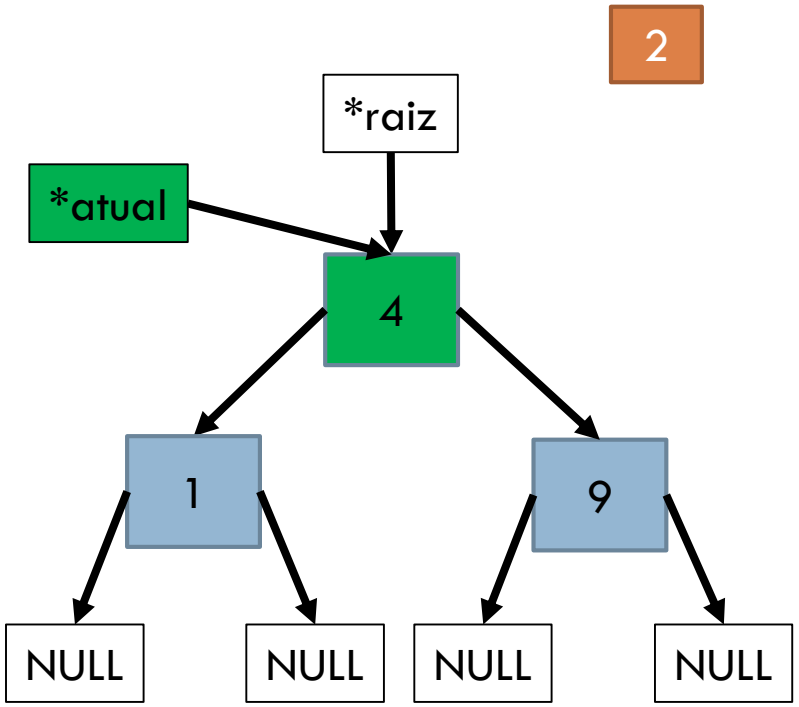
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



Busca em ABB

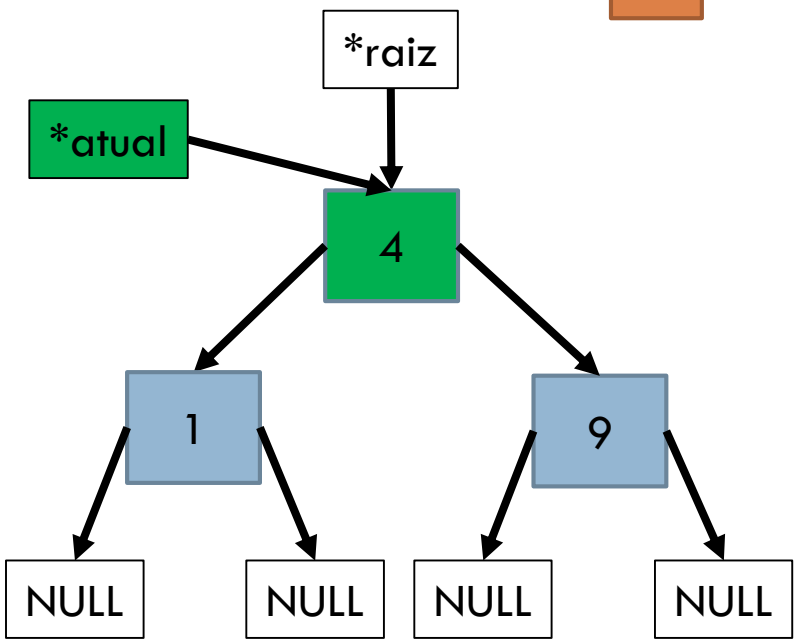

```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



Busca em ABB

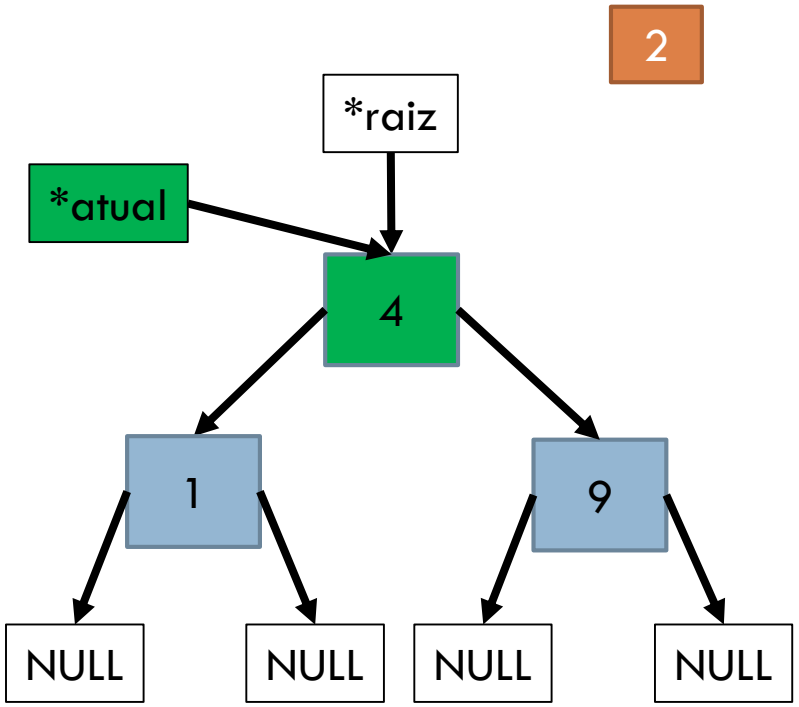

```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
     ➡if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```

2



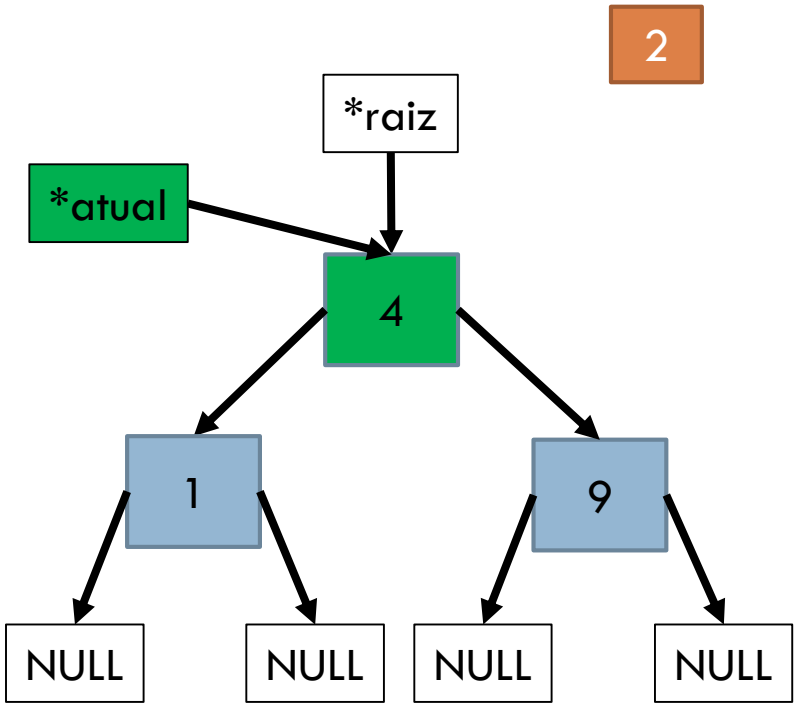
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



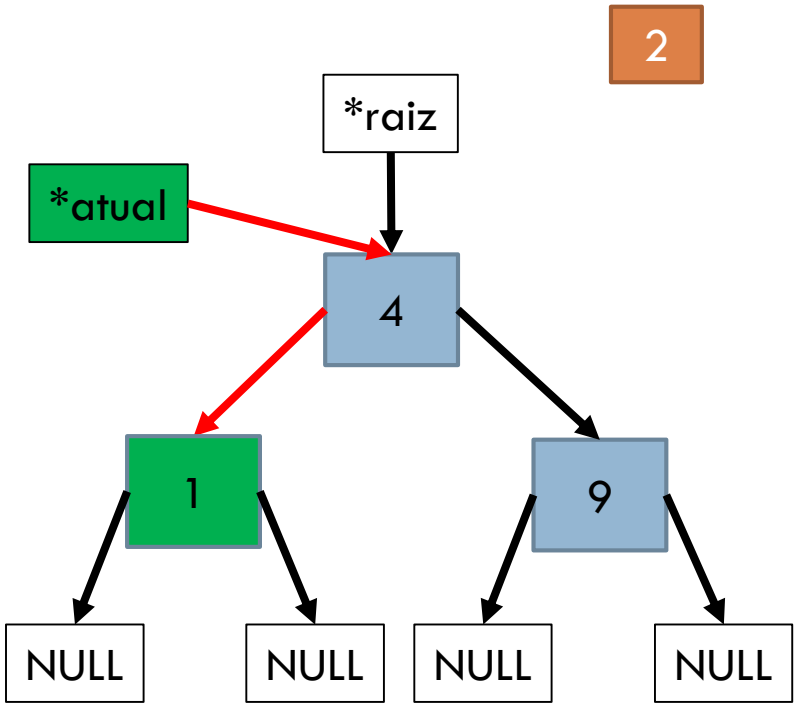
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



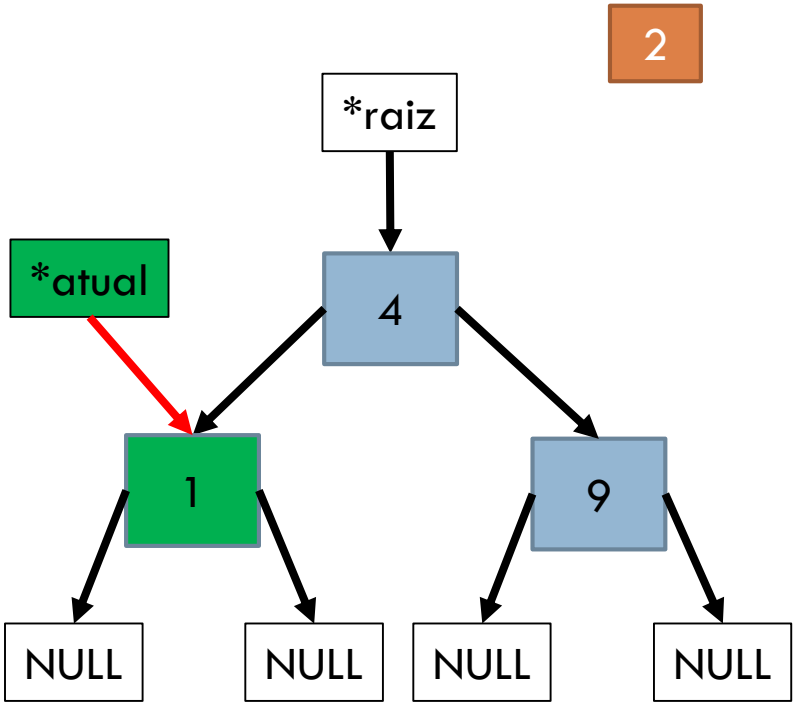
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



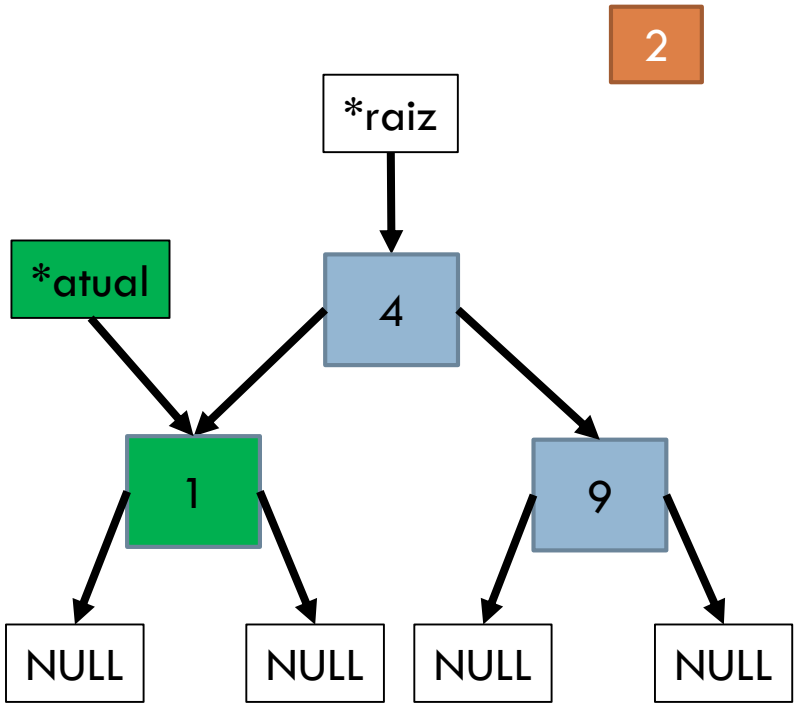
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



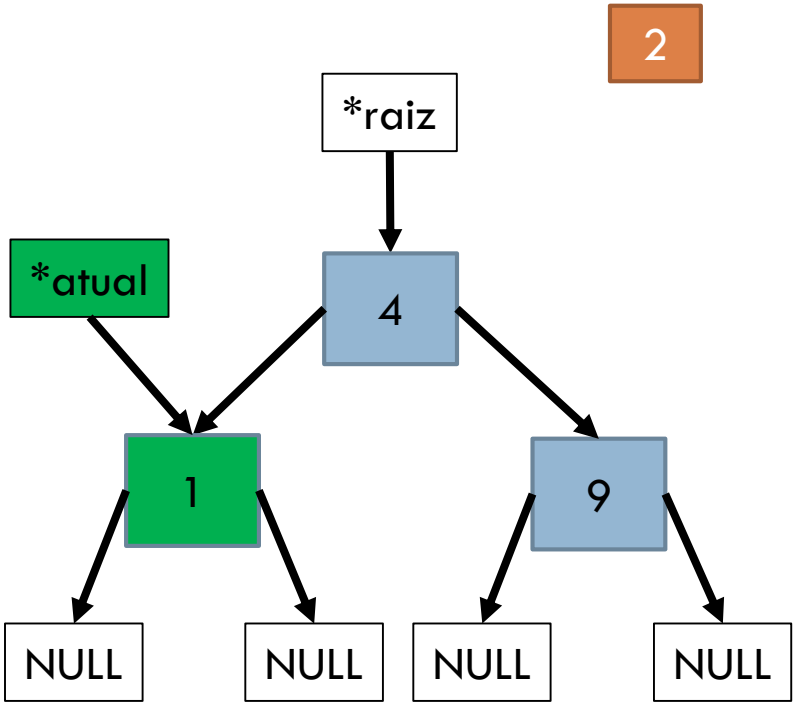
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



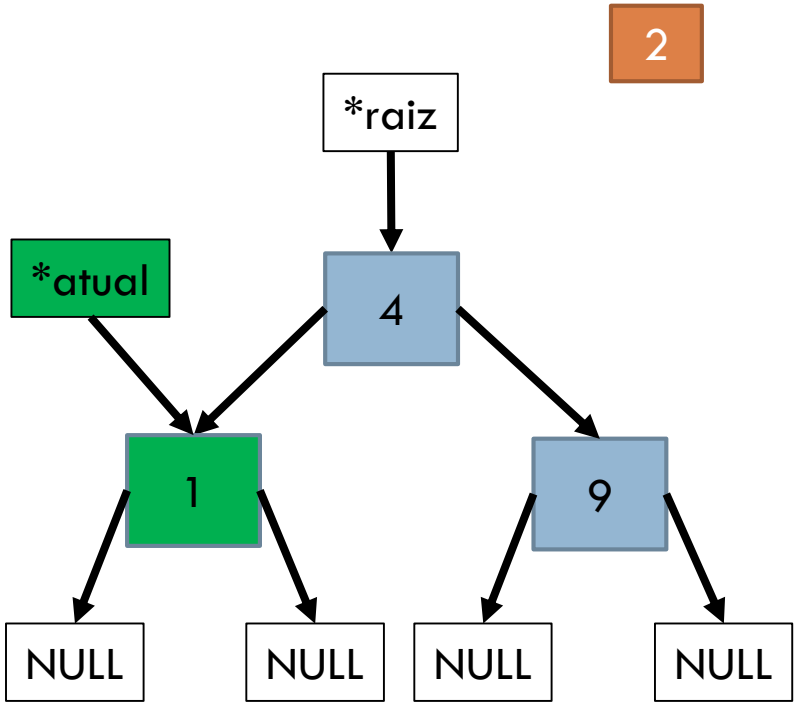
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



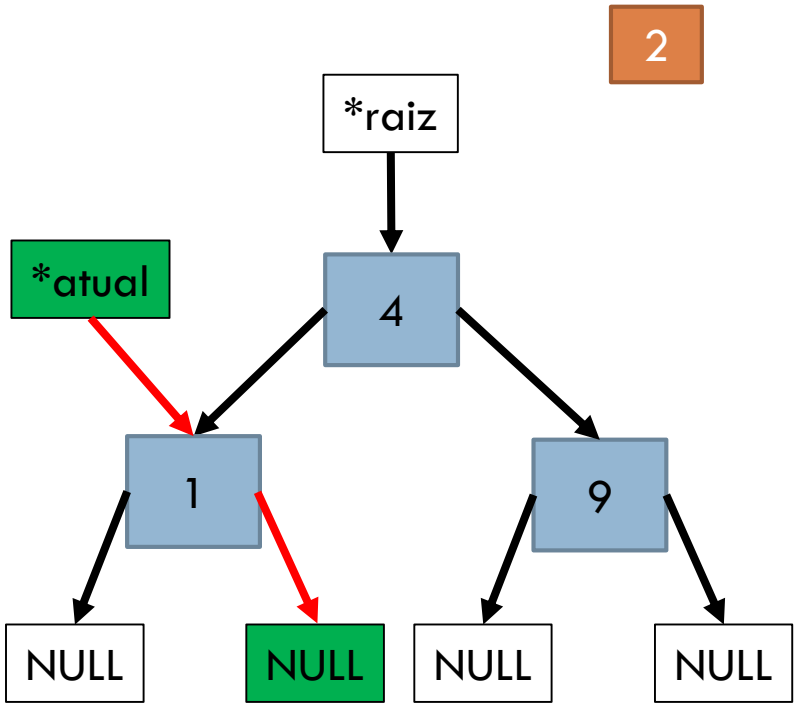
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



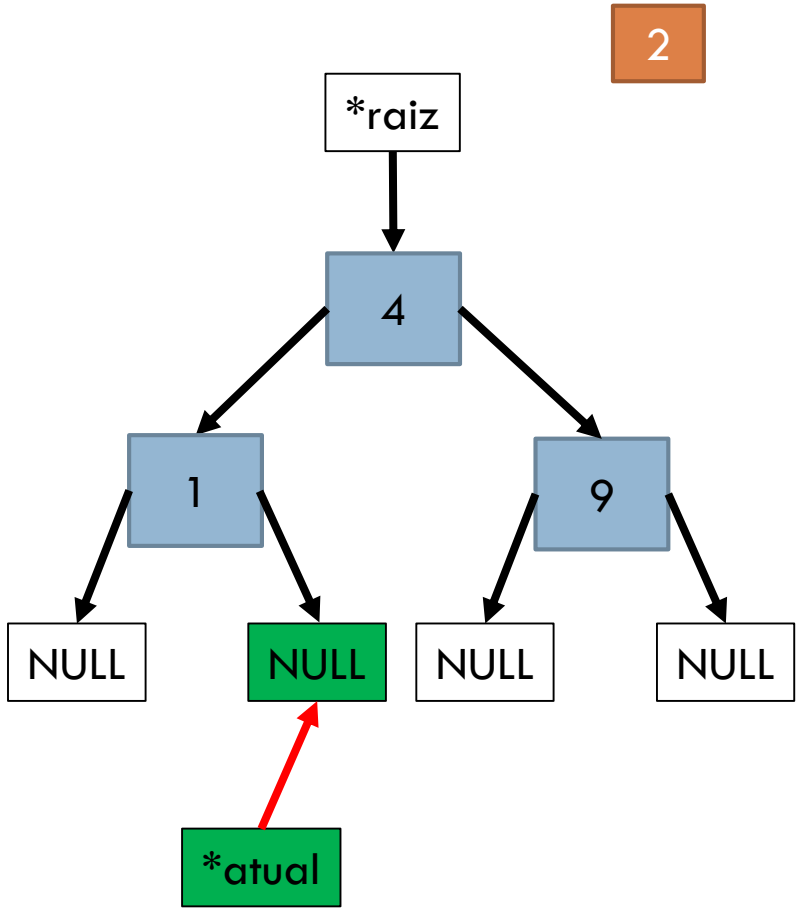
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
         ➔ atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



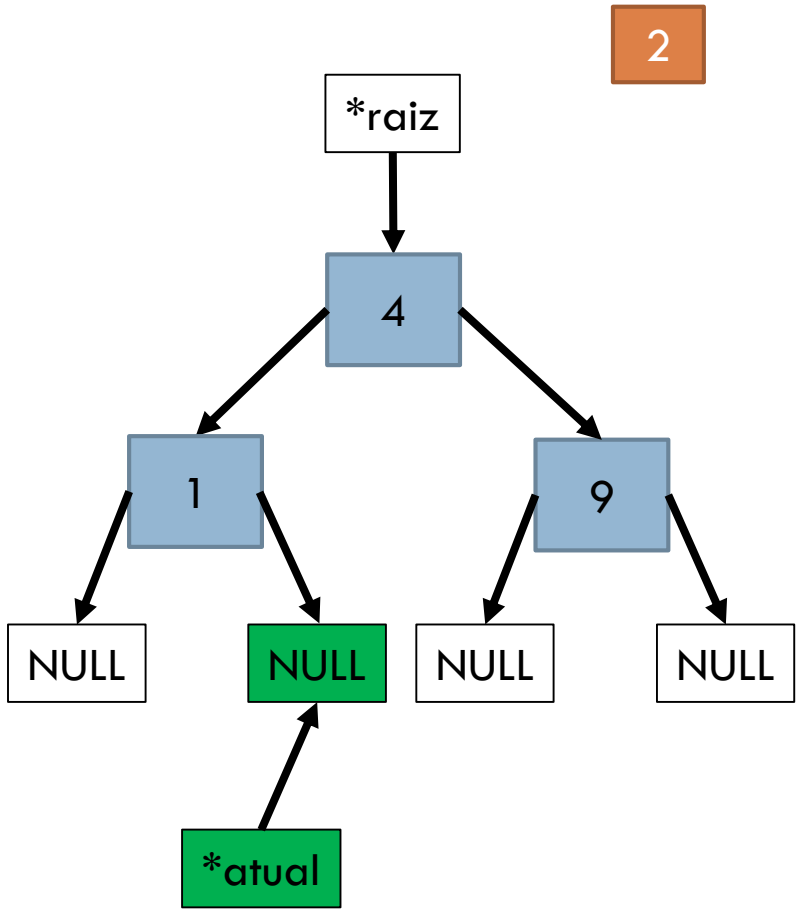
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



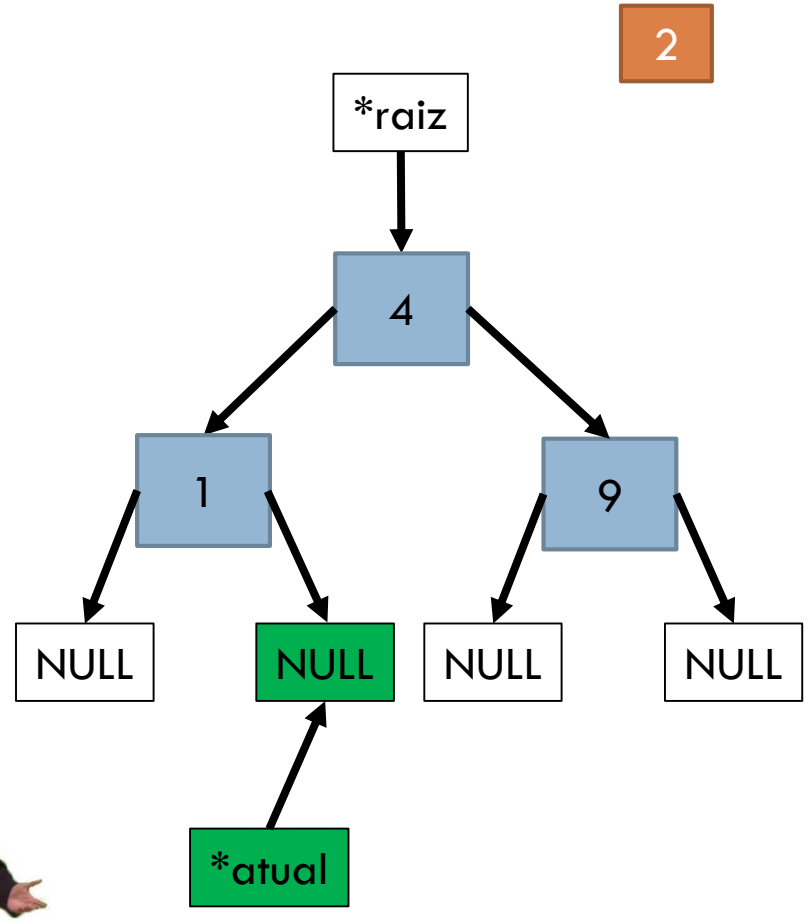
Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



Busca em ABB


```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return NULL;
}
```



Exercícios

- Faça a inserção dos dados:
  - 6-3-2-1-4-5-9-8-7-10-11
  - 1-2-3-4-5-6-7-8-9-10-11
- Realize o percurso Em-Ordem nas árvores anteriores.
- Quantos deslocamentos são necessários para encontrar o nó 11?

Remoção em ABB

- Receber a raiz de uma árvore e uma chave.
- Busca pela chave na árvore.
  - Verificar se a chave é menor que a raiz:
    - Descer para a subárvore esquerda;
  - Verificar se a chave é maior que a raiz:
    - Descer para a subárvore direita;
- Realizar **escolha do substituto** para o dado que será removido.
- Substituir o dado e realizar a remoção.

Remoção em ABB

- Algoritmo remove(*raiz, chave):
  - Se posição está vazia (NULL), não existe;
  - Se é menor que raiz, remove((*raiz)->**esq**, **chave**);
  - Se é maior que raiz, remove((*raiz)->**dir**, **chave**);
  - Se é igual, encontra substituto:

Remoção em ABB

- Algoritmo remove(*raiz, chave):
  - Se posição está vazia (NULL), não existe;
  - Se é menor que raiz, remove((*raiz)->**esq**, **chave**);
  - Se é maior que raiz, remove((*raiz)->**dir**, **chave**);
  - Se é igual, encontra substituto:
    - Caso 1: se é **folha**, apenas remove;

Remoção em ABB

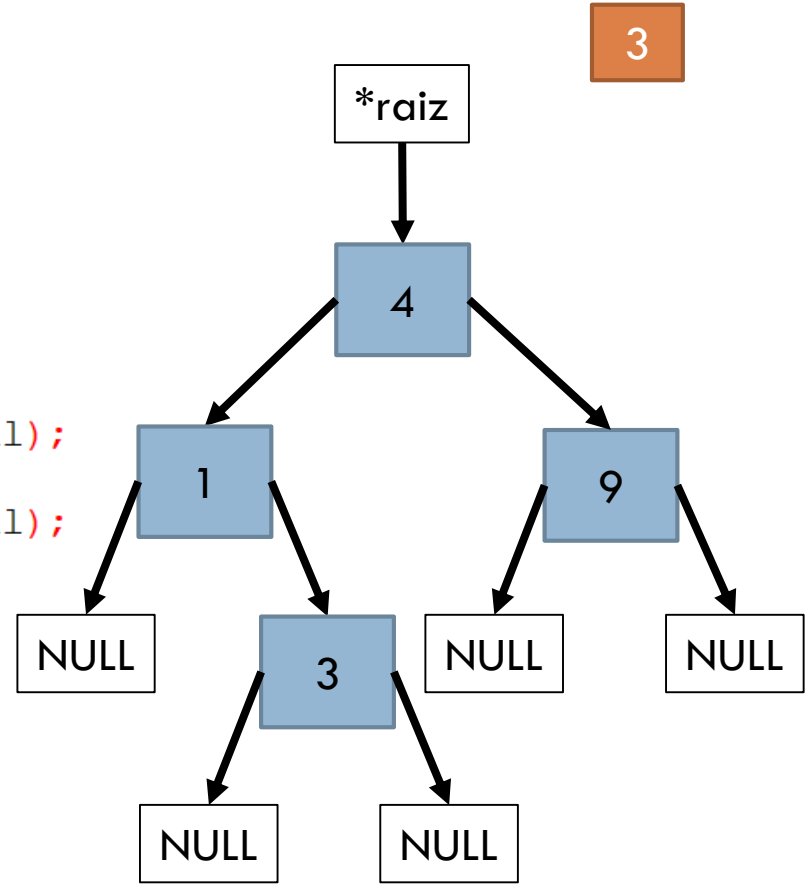
- Algoritmo remove(*raiz, chave):
  - Se posição está vazia (NULL), não existe;
  - Se é menor que raiz, remove((*raiz)->**esq**, **chave**);
  - Se é maior que raiz, remove((*raiz)->**dir**, **chave**);
  - Se é igual, encontra substituto:
    - Caso 1: se é **folha**, apenas remove;
    - Caso 2: se possui **apenas uma subárvore** (esquerda ou direita), a raiz da subárvore substitui o nó.

Remoção em ABB

- Algoritmo remove(*raiz, chave):
  - Se posição está vazia (NULL), não existe;
  - Se é menor que raiz, remove((*raiz)->**esq**, **chave**);
  - Se é maior que raiz, remove((*raiz)->**dir**, **chave**);
  - Se é igual, encontra substituto:
    - Caso 1: se é **folha**, apenas remove;
    - Caso 2: se possui **apenas uma subárvore** (esquerda ou direita), a raiz da subárvore substitui o nó.
    - Caso 3: se possui **duas subárvores**:
      - Substitui pelo menor nó da direita; ou
      - Substitui pelo maior nó da esquerda.

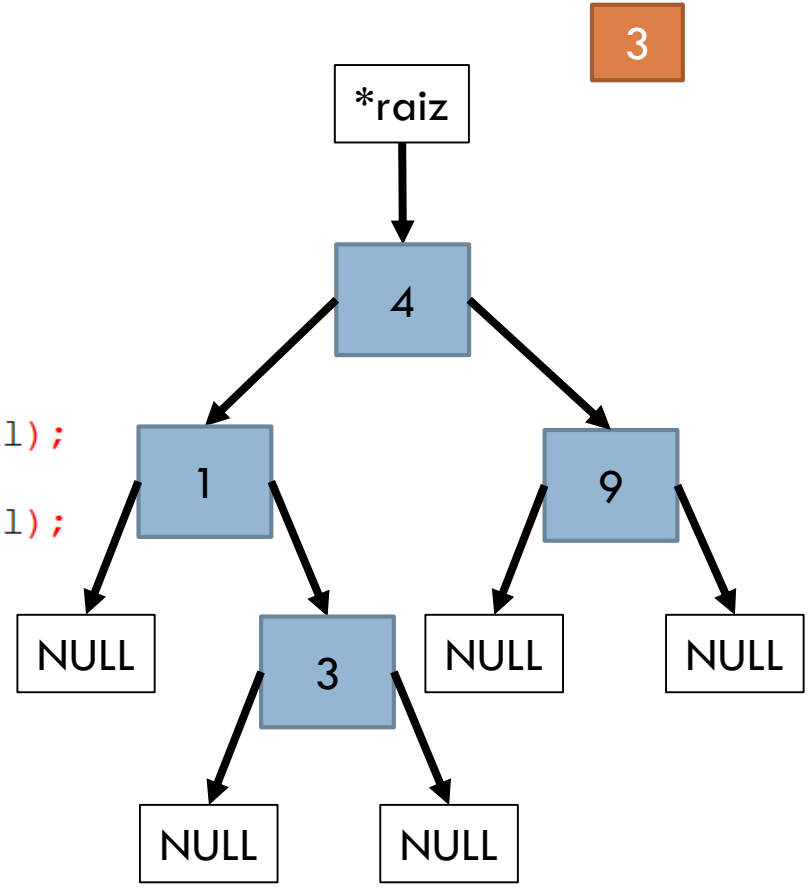
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



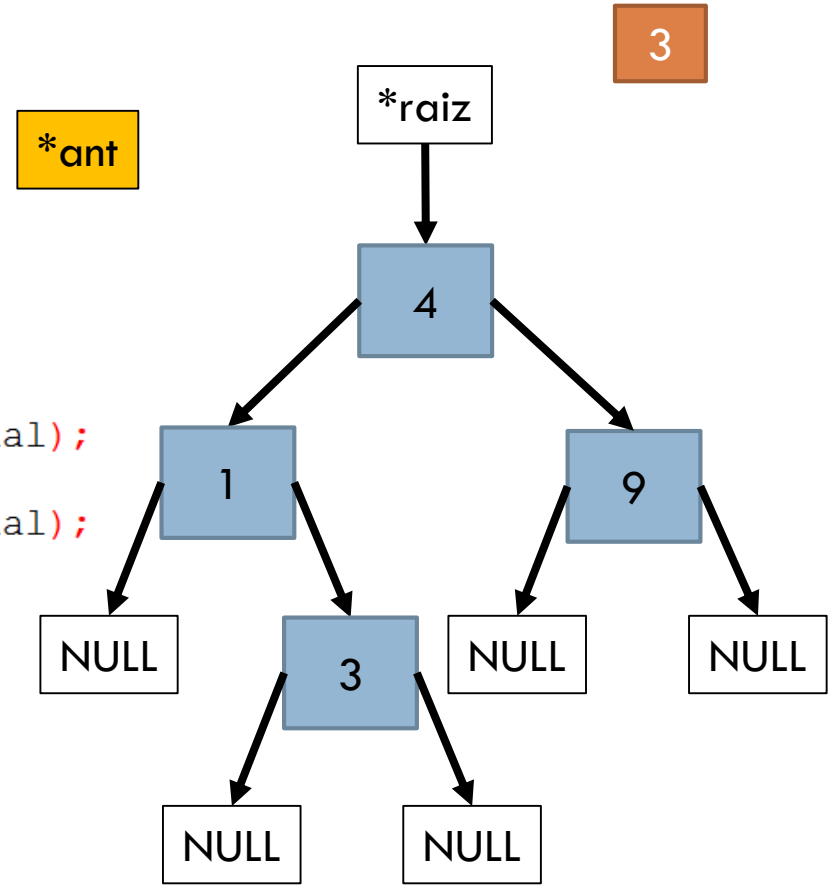
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



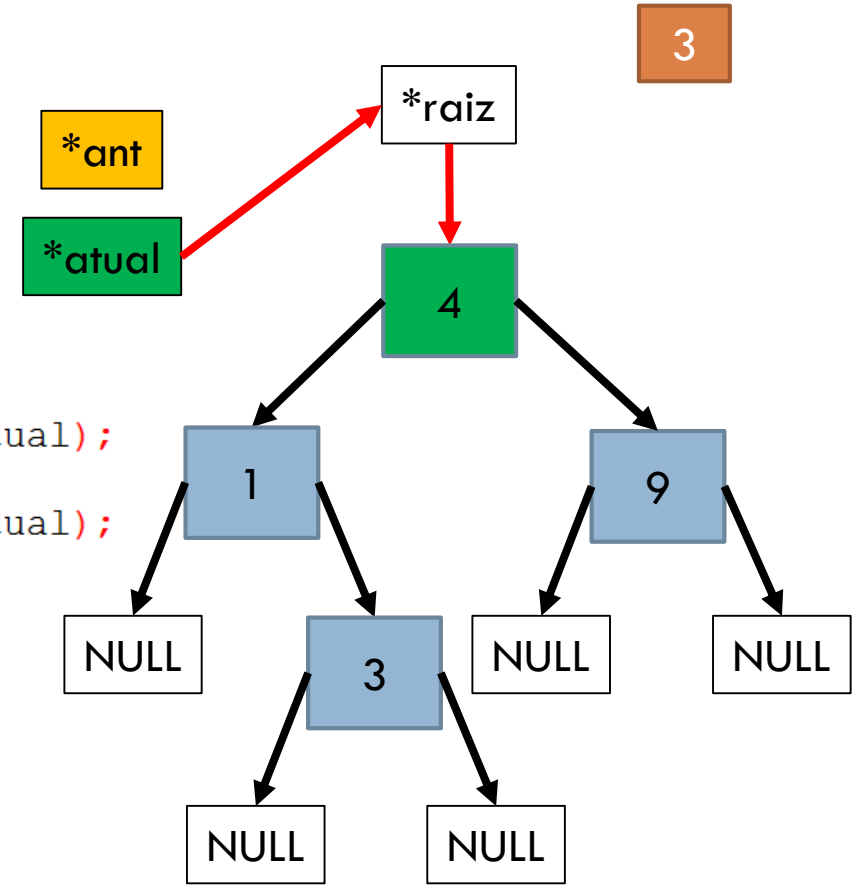
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



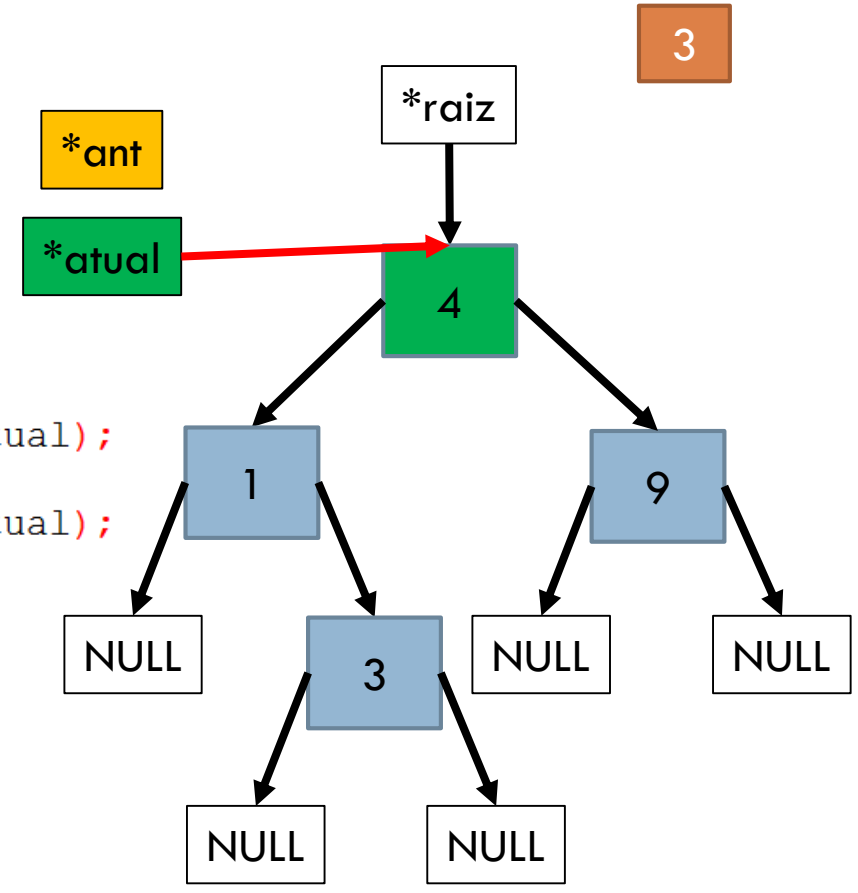
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



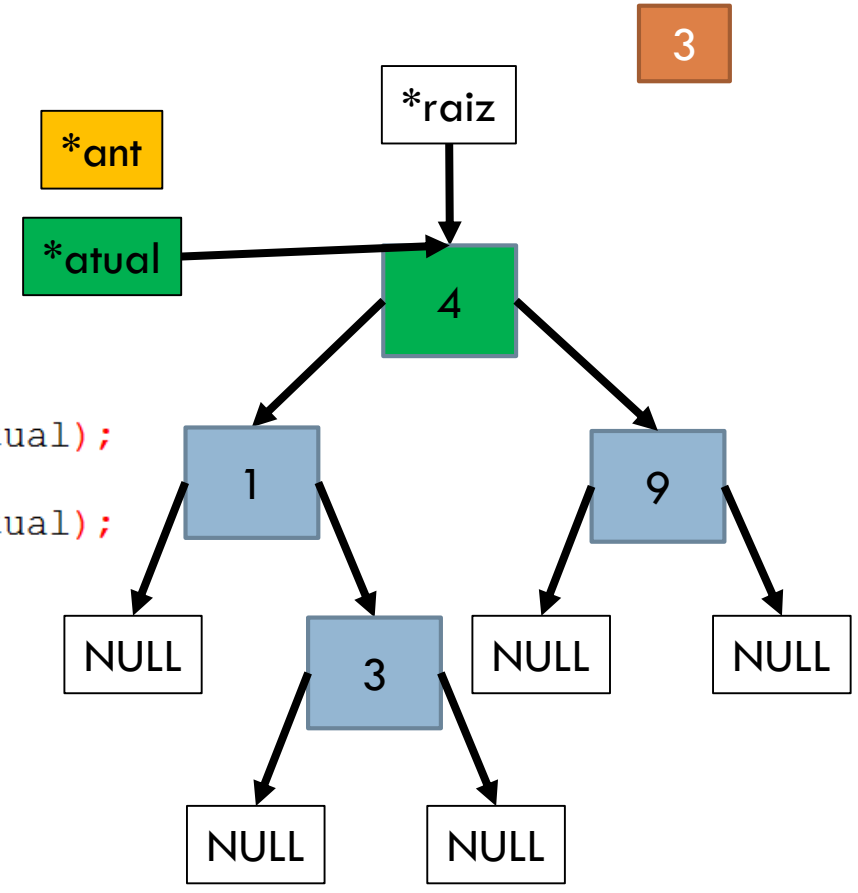
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



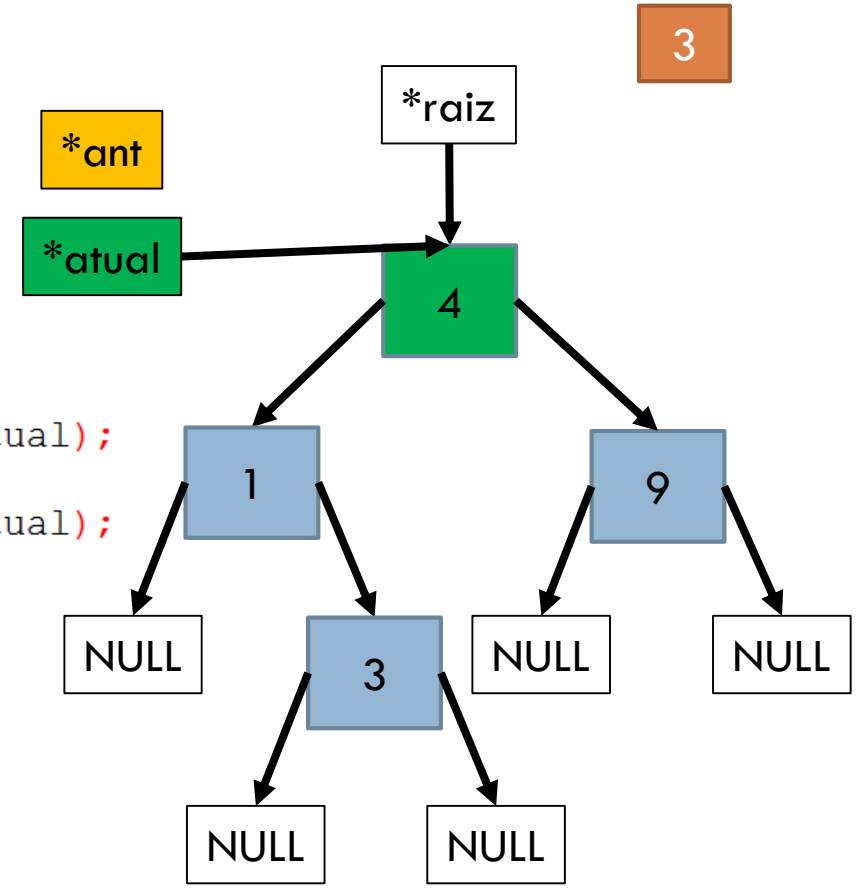
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



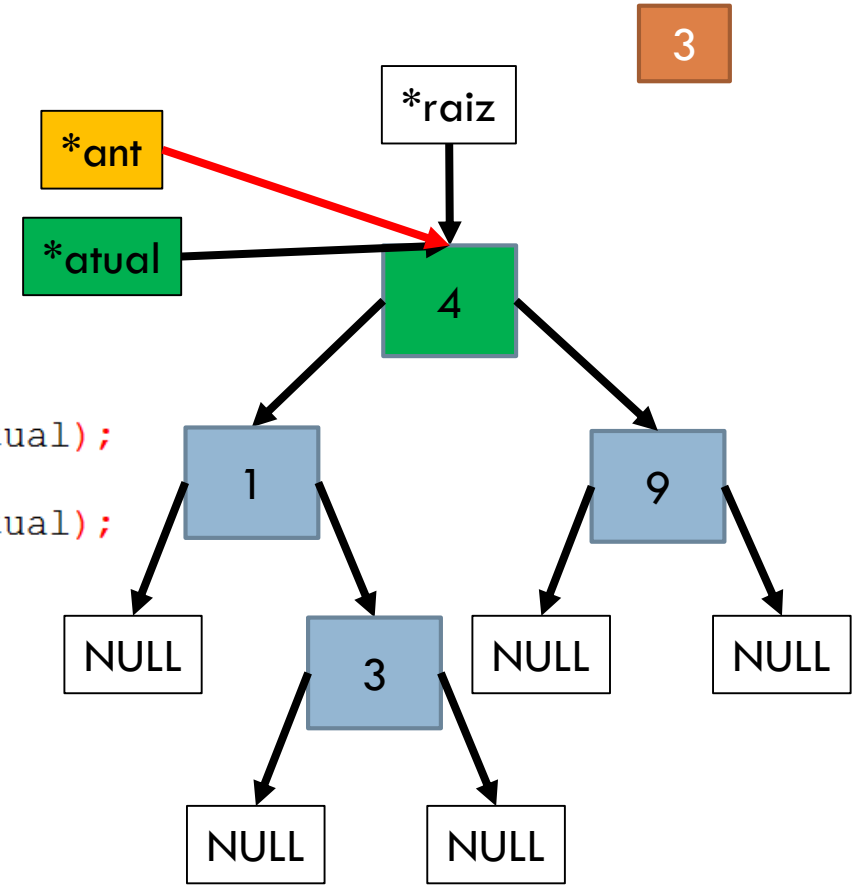
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



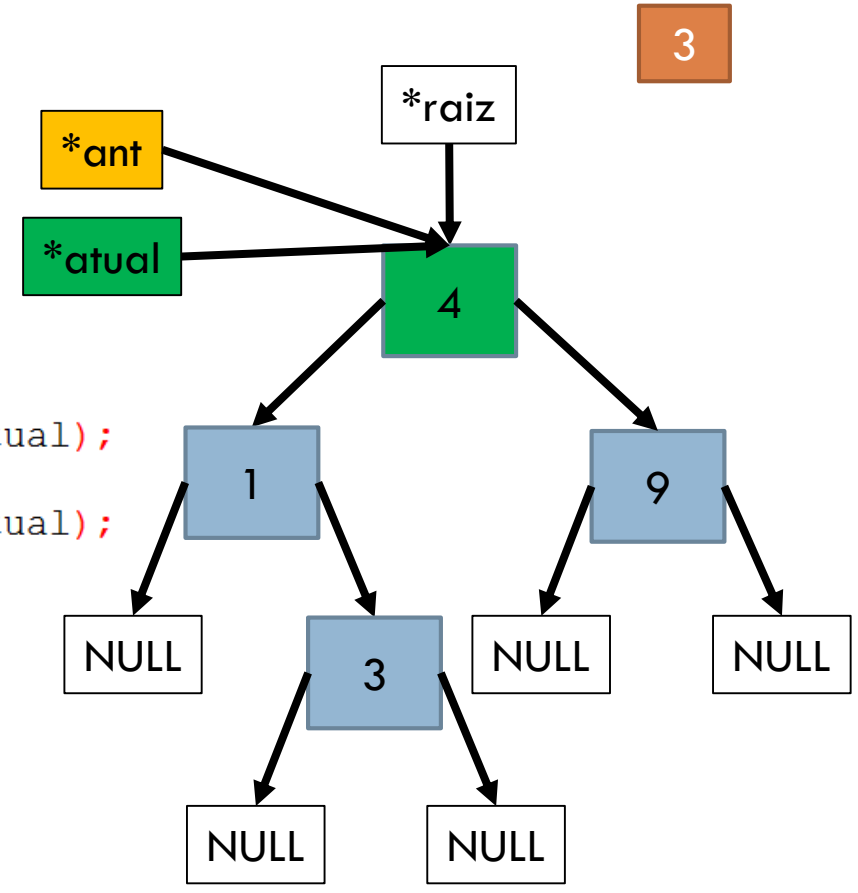
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



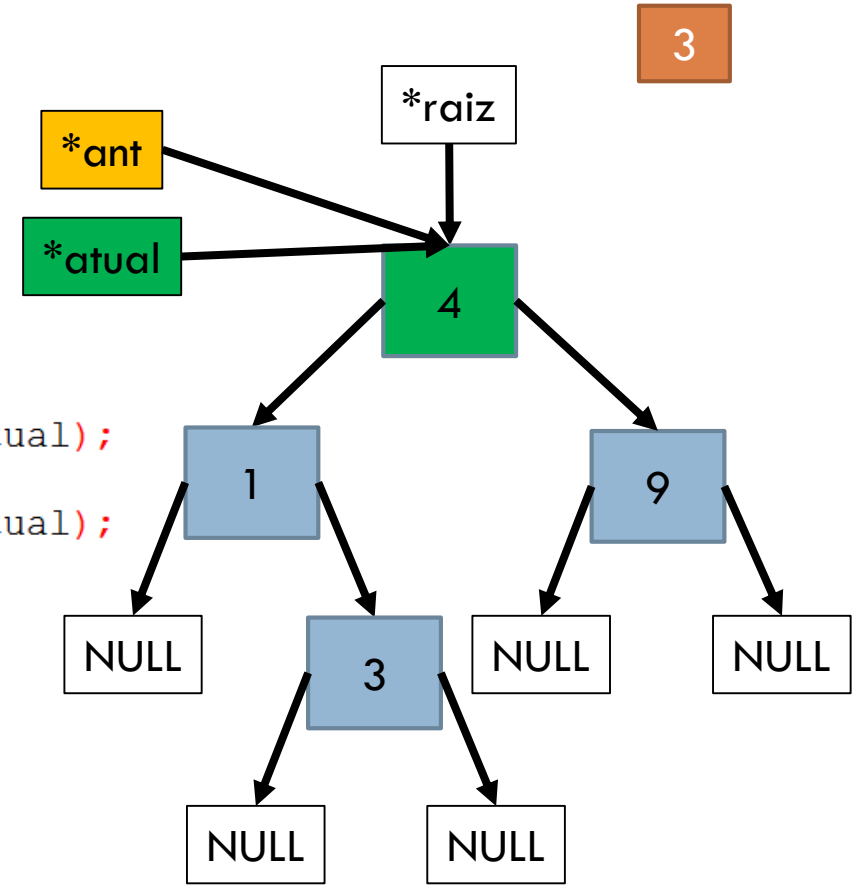
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



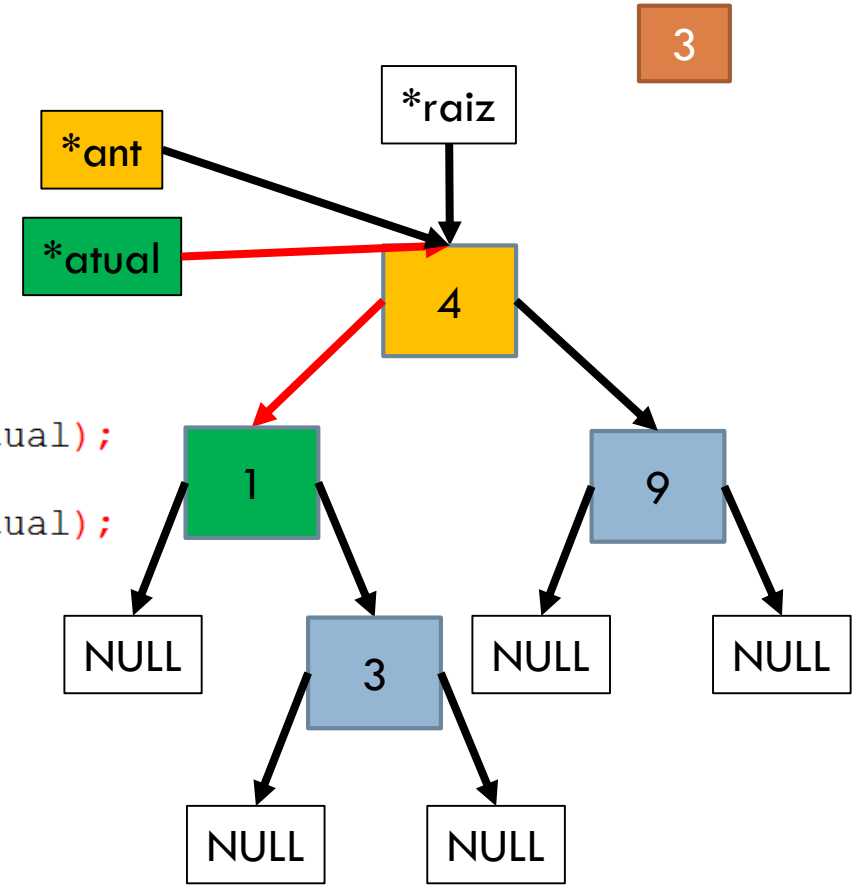
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



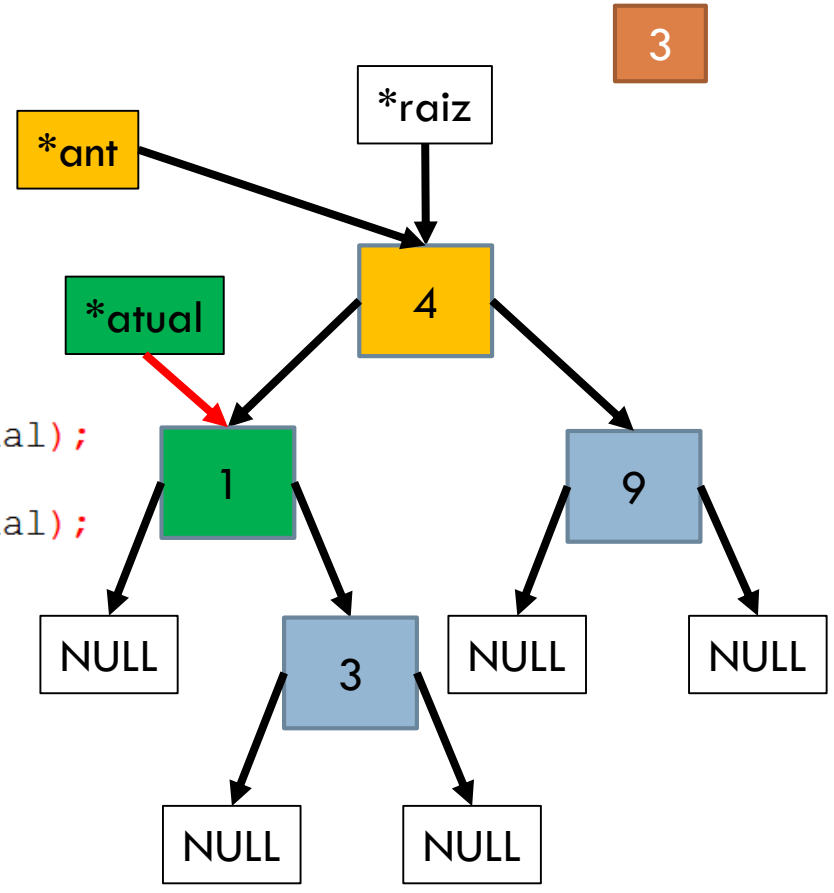
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



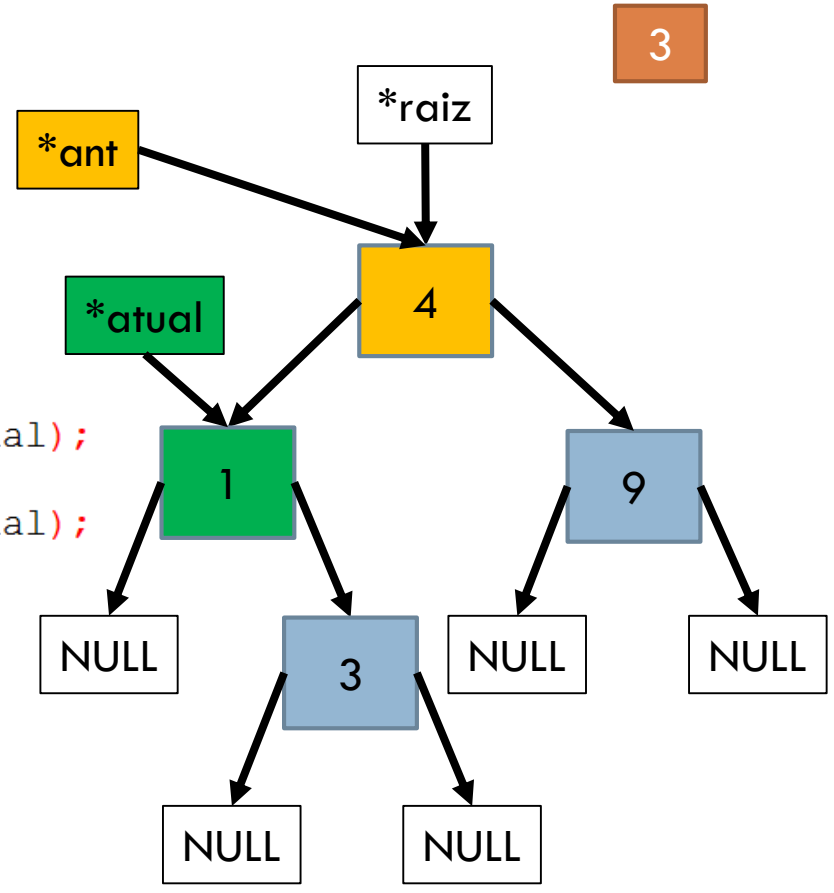
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



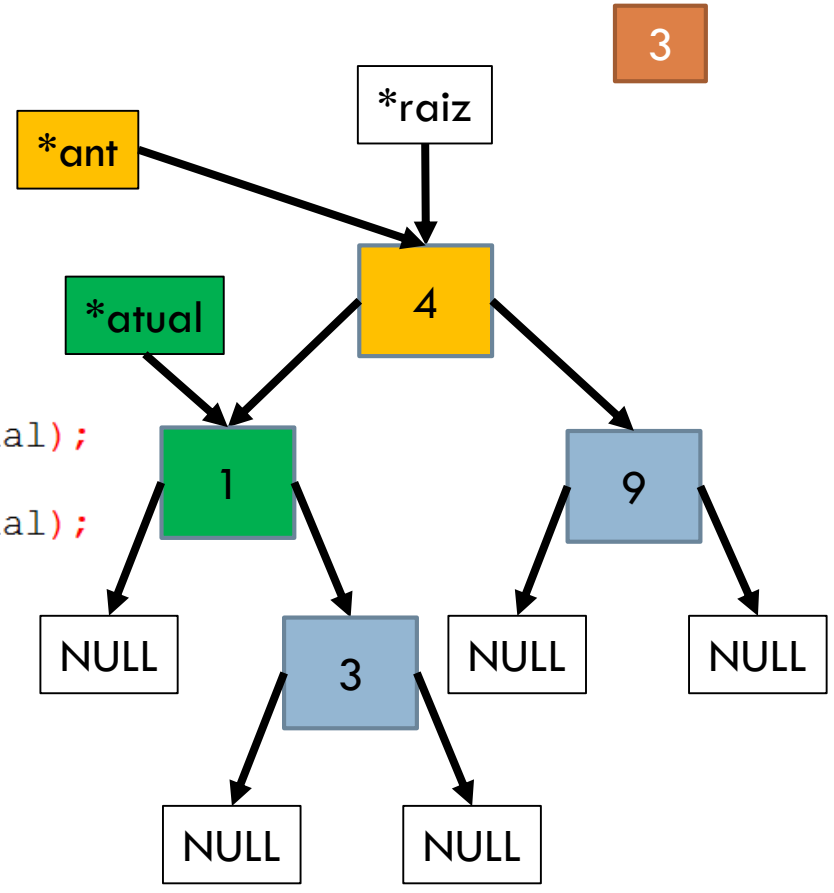
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



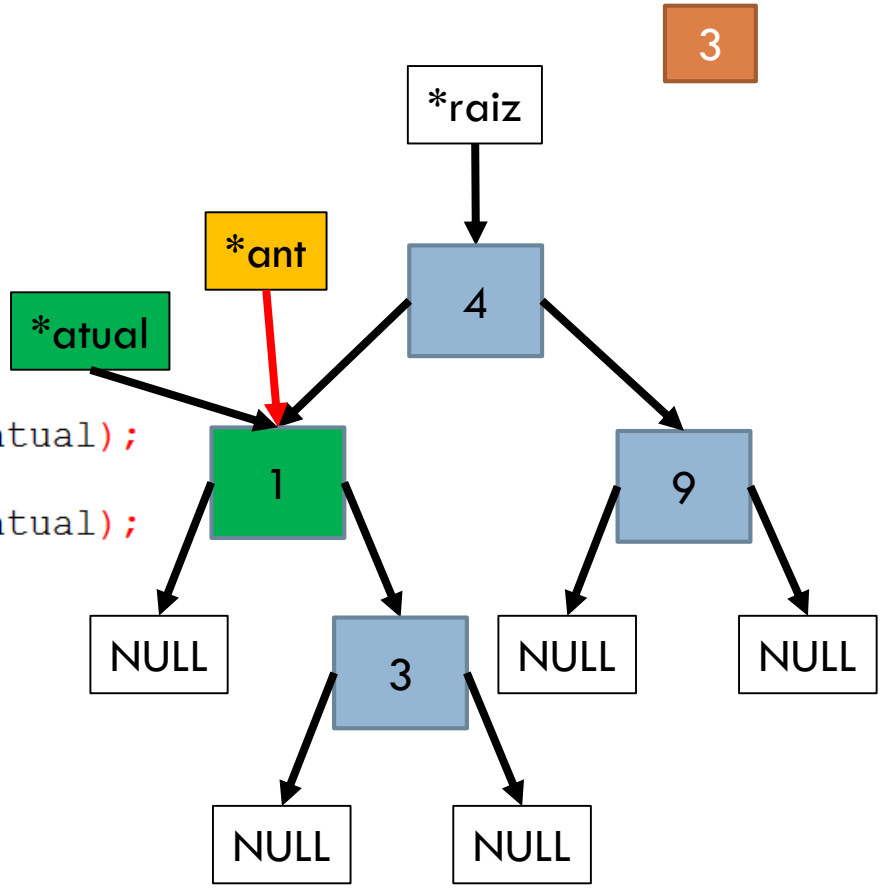
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



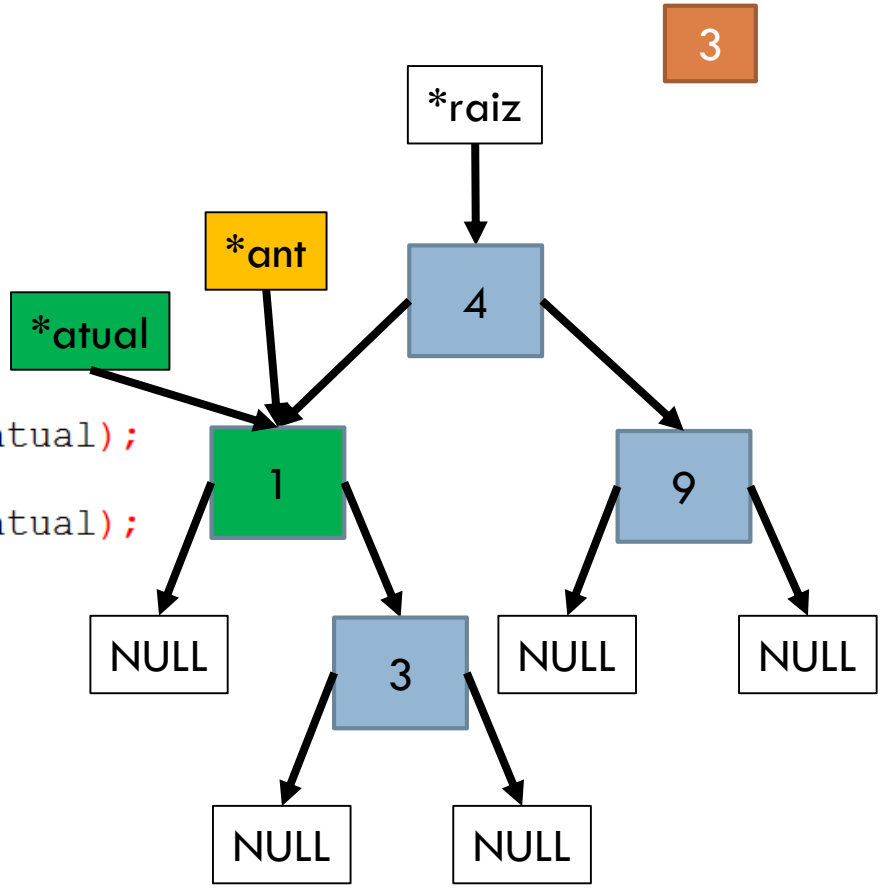
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



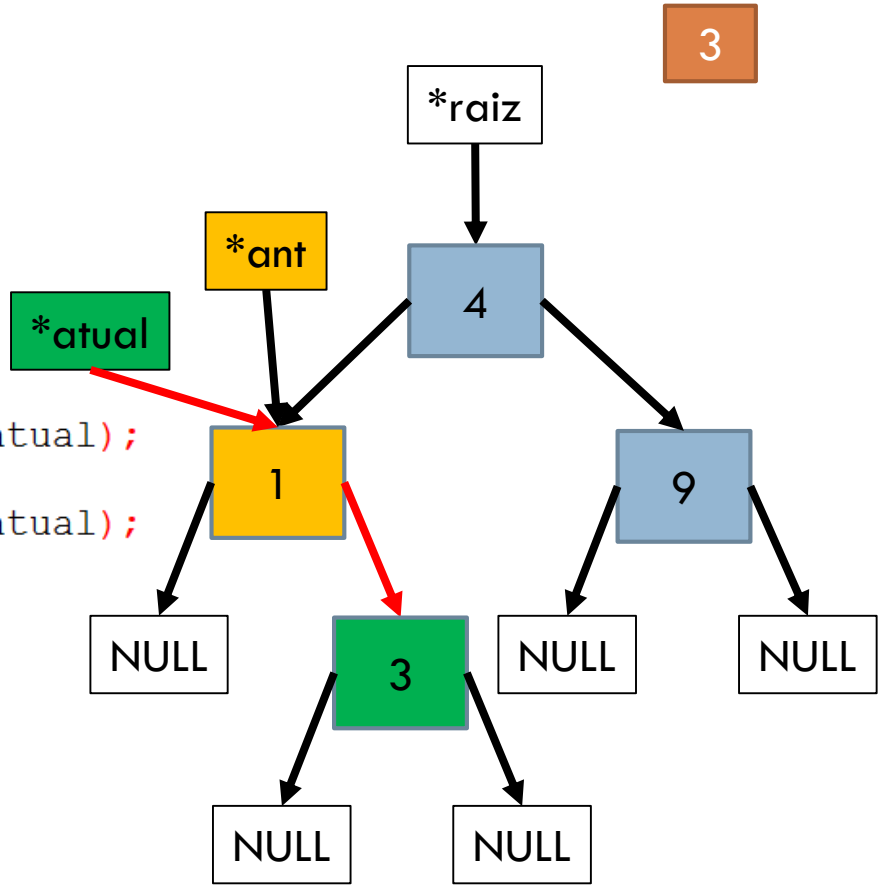
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



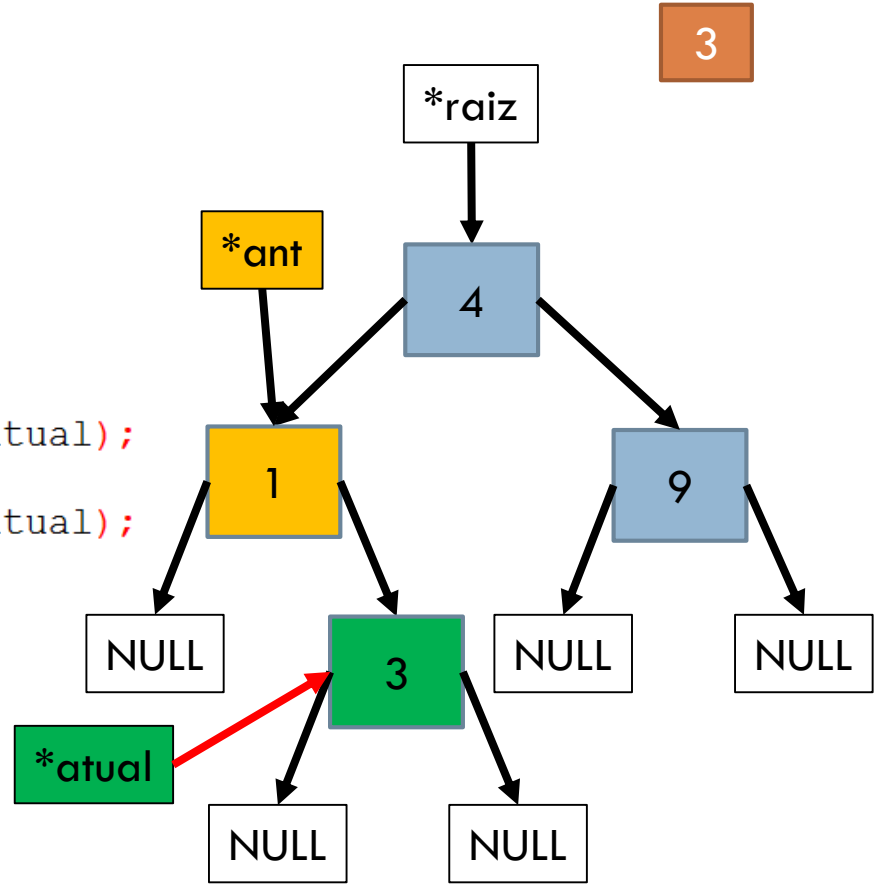
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



Remoção em ABB

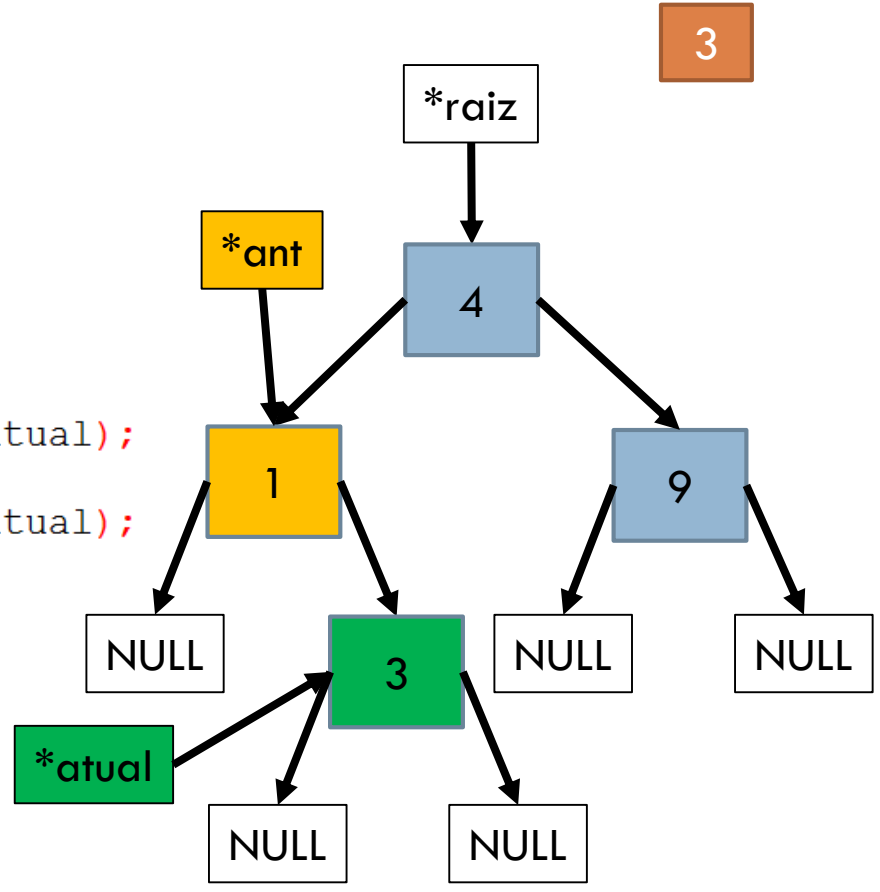

```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



Remoção em ABB

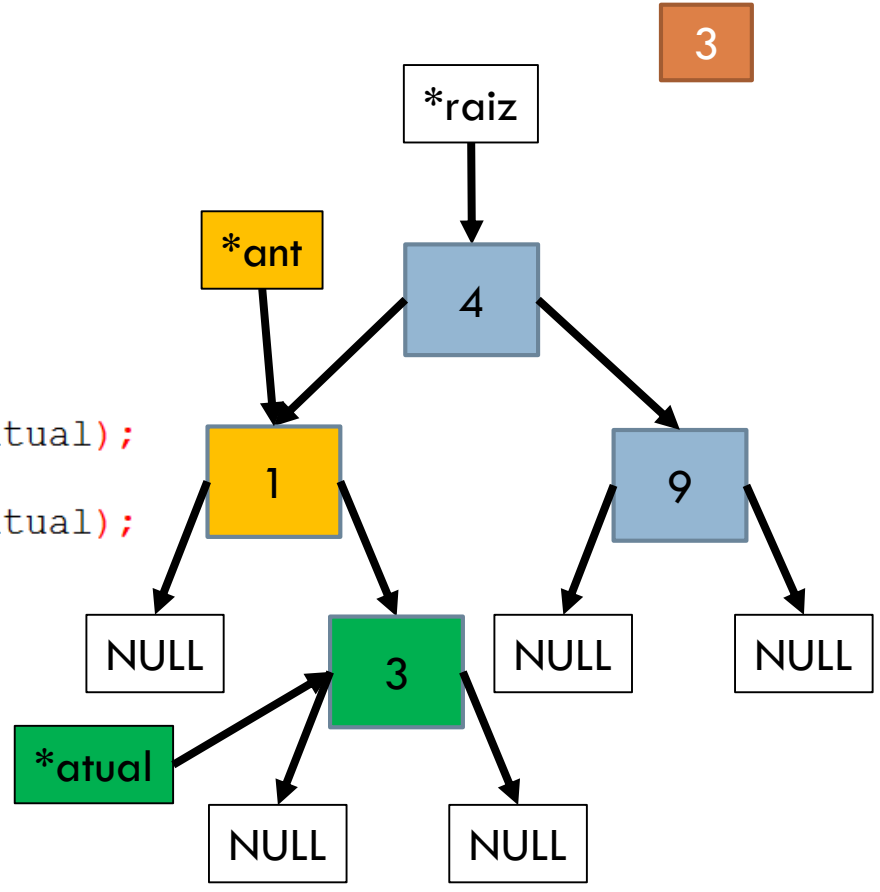
150

```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



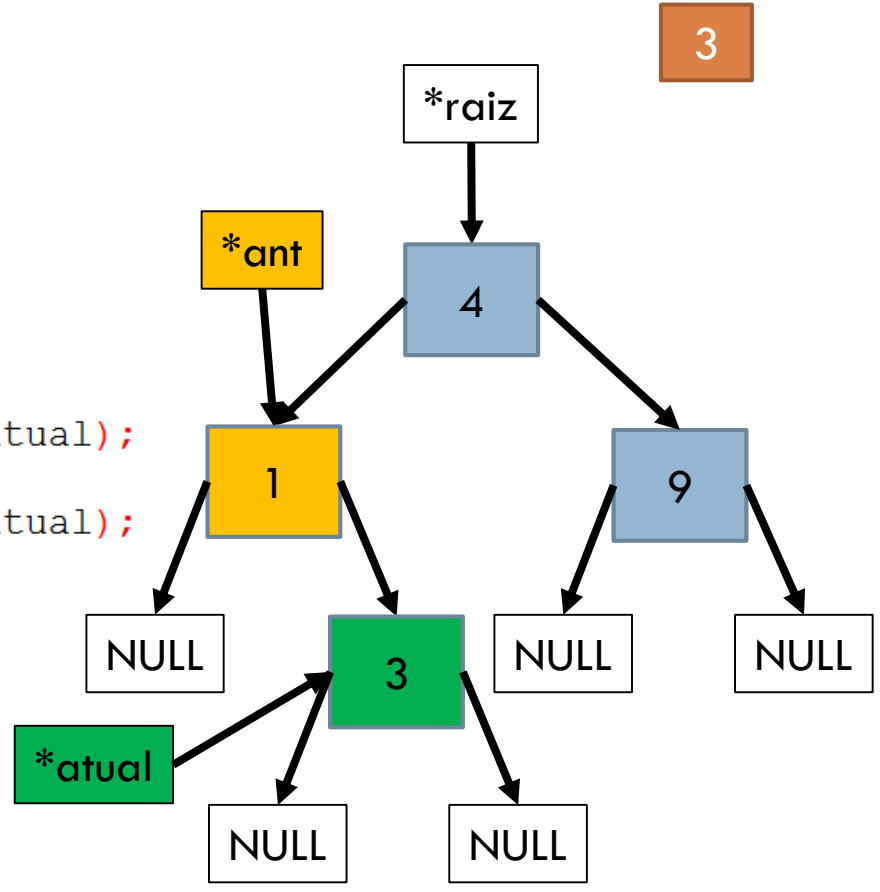
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
    ➡ if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



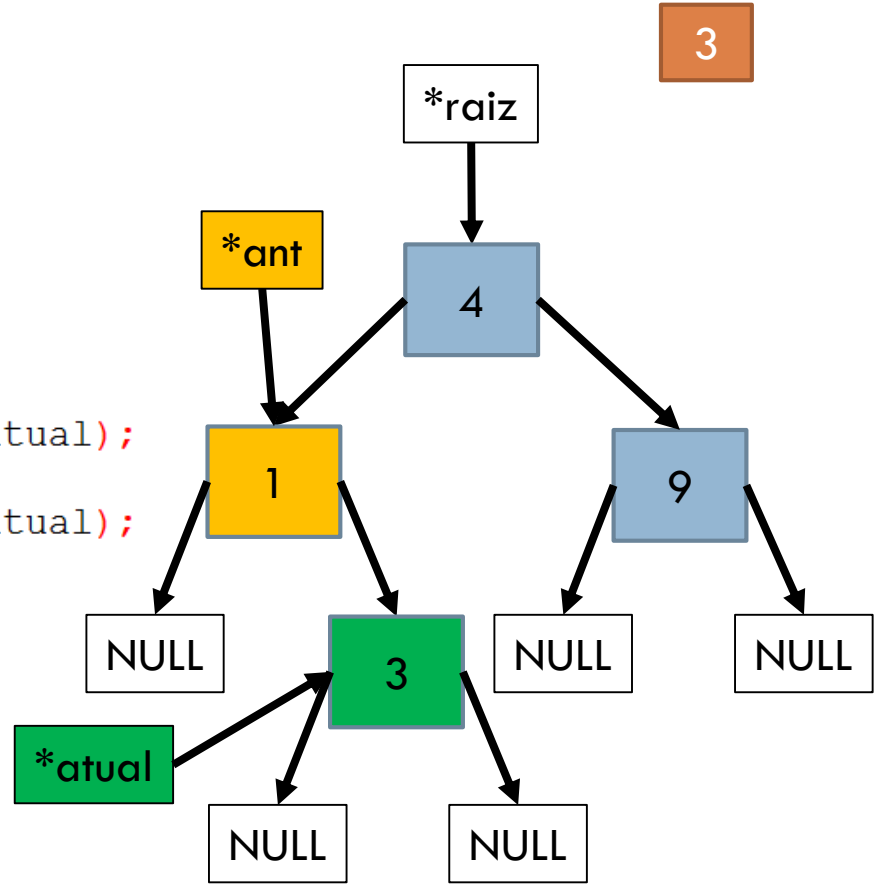
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



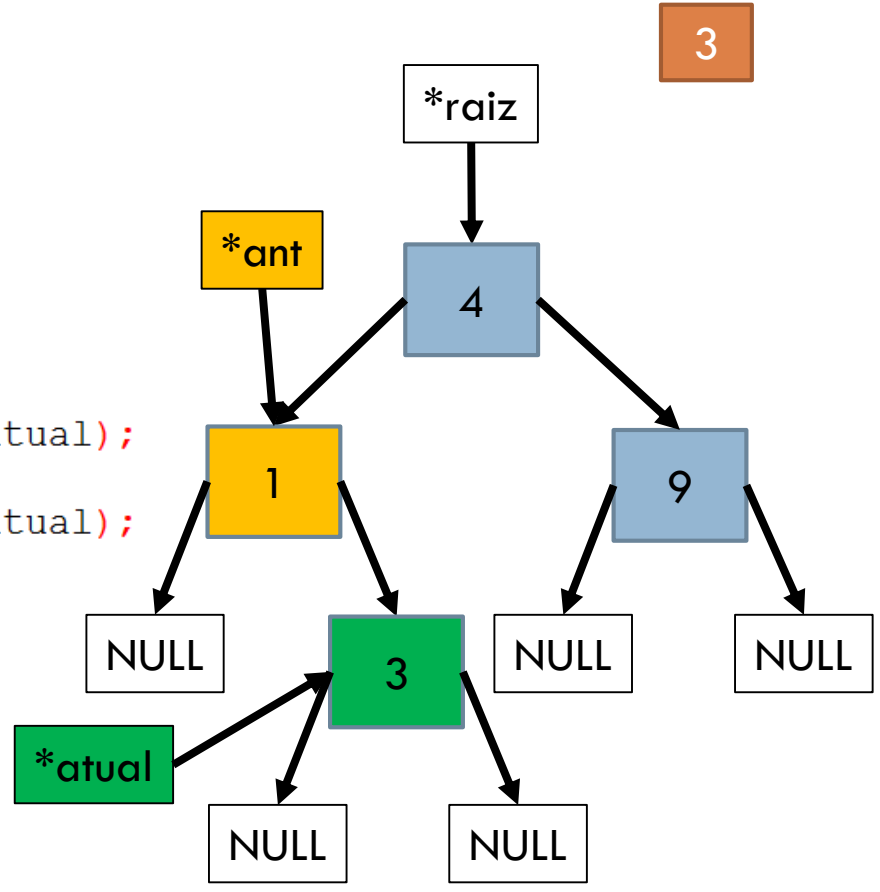
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



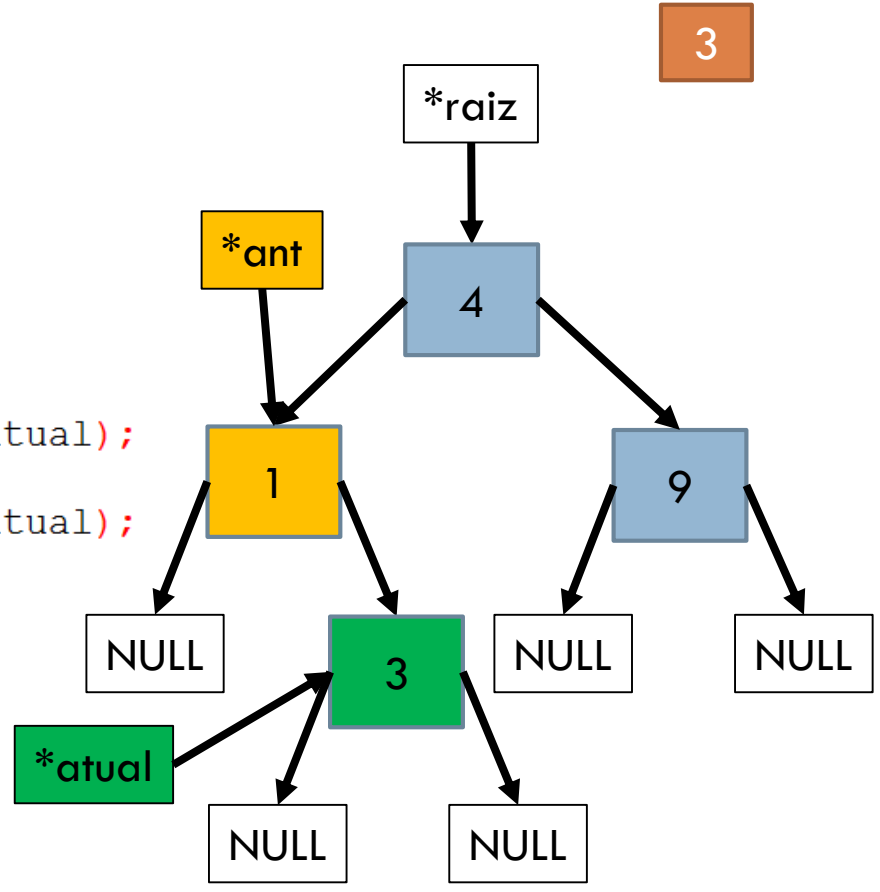
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



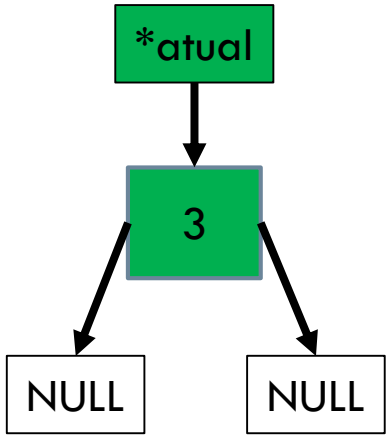
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



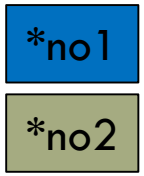
Remoção em ABB

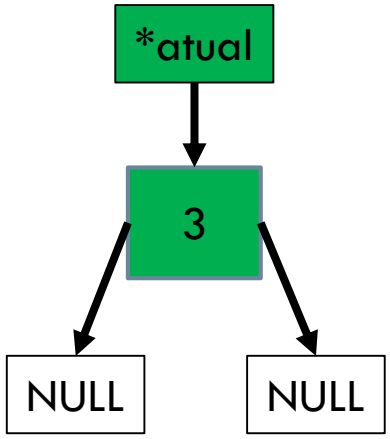

```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

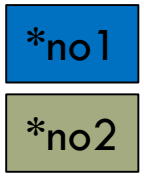

```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

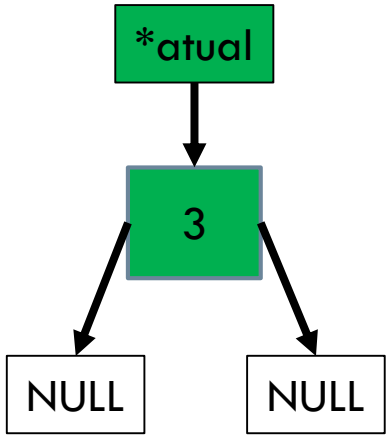




Remoção em ABB

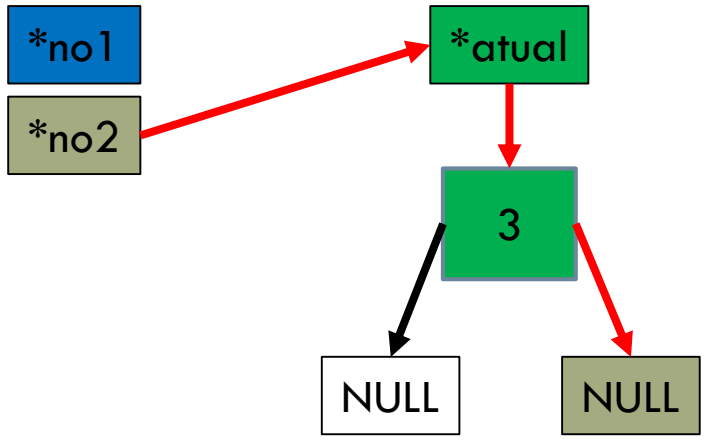

```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```





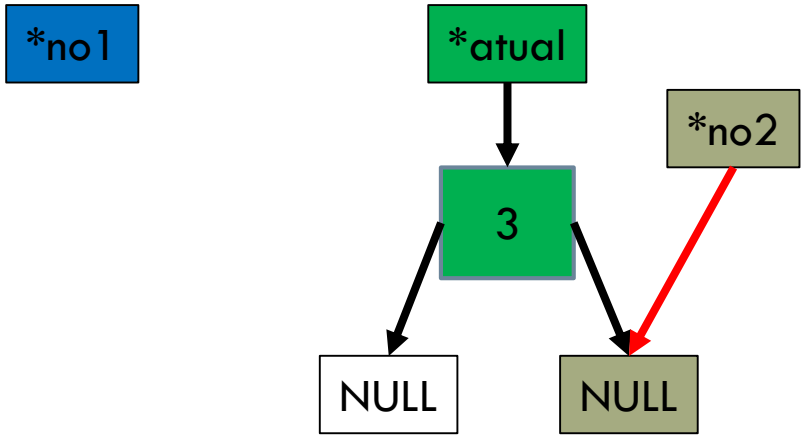
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



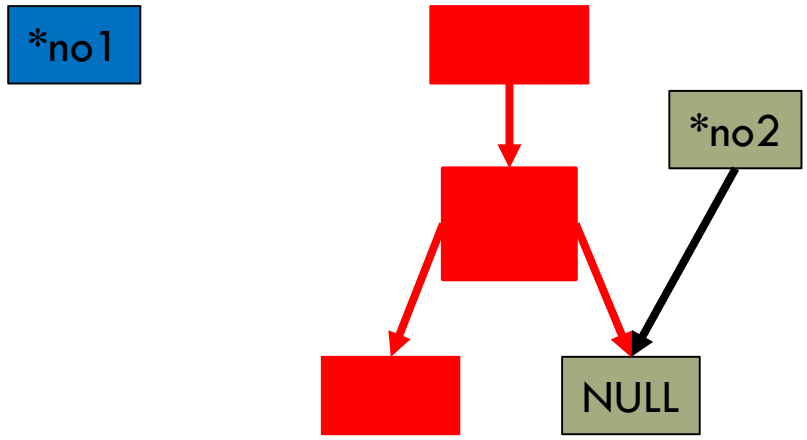
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



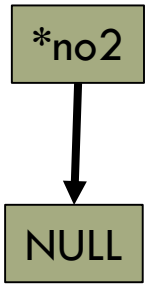
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



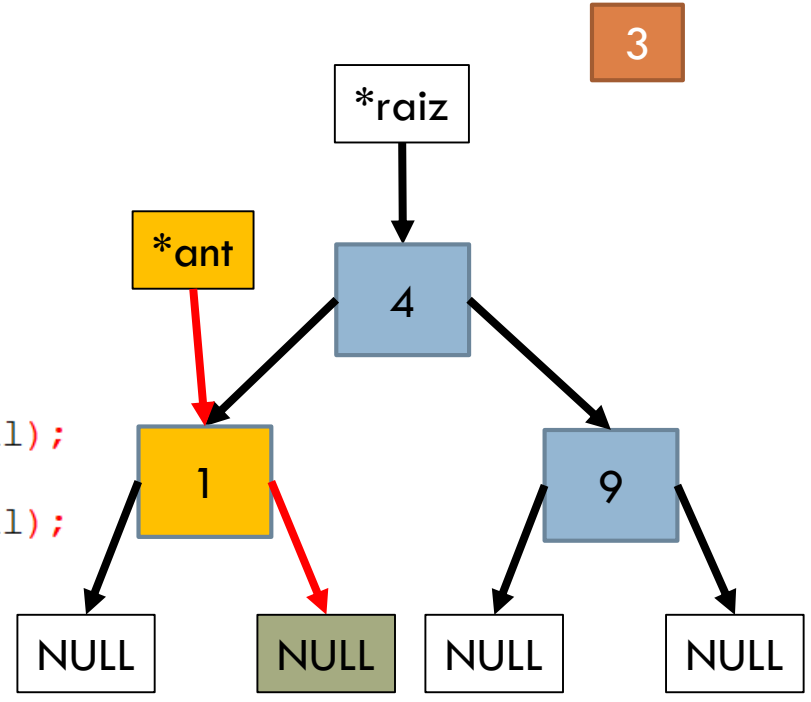
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



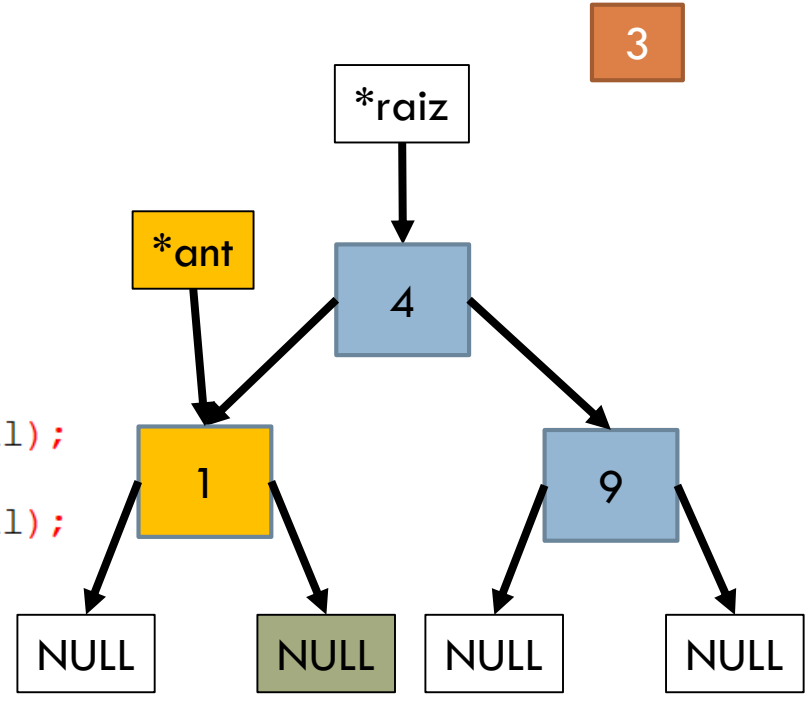
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```

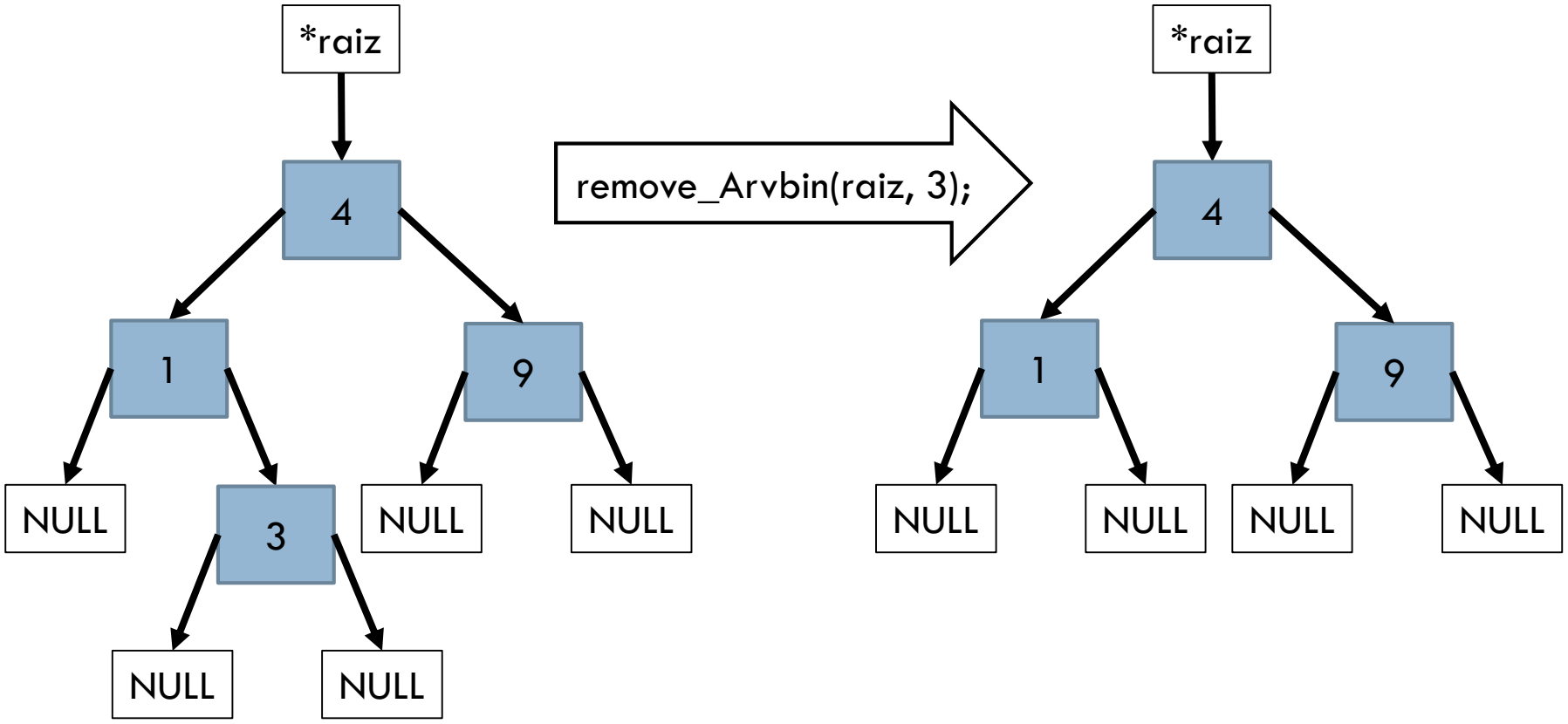


Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```

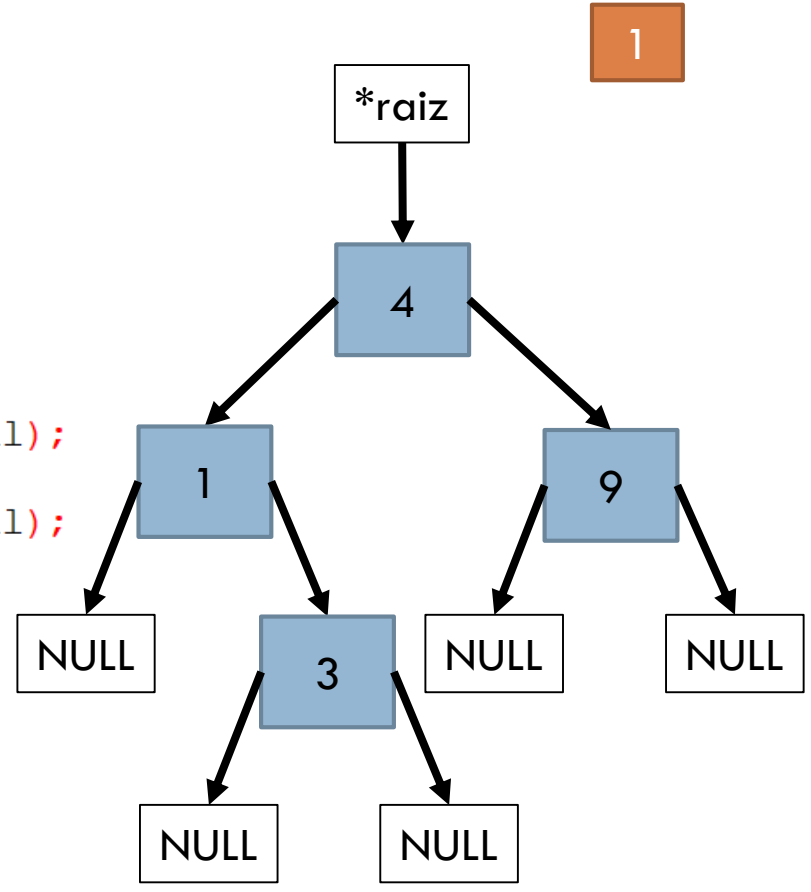


Remoção em ABB



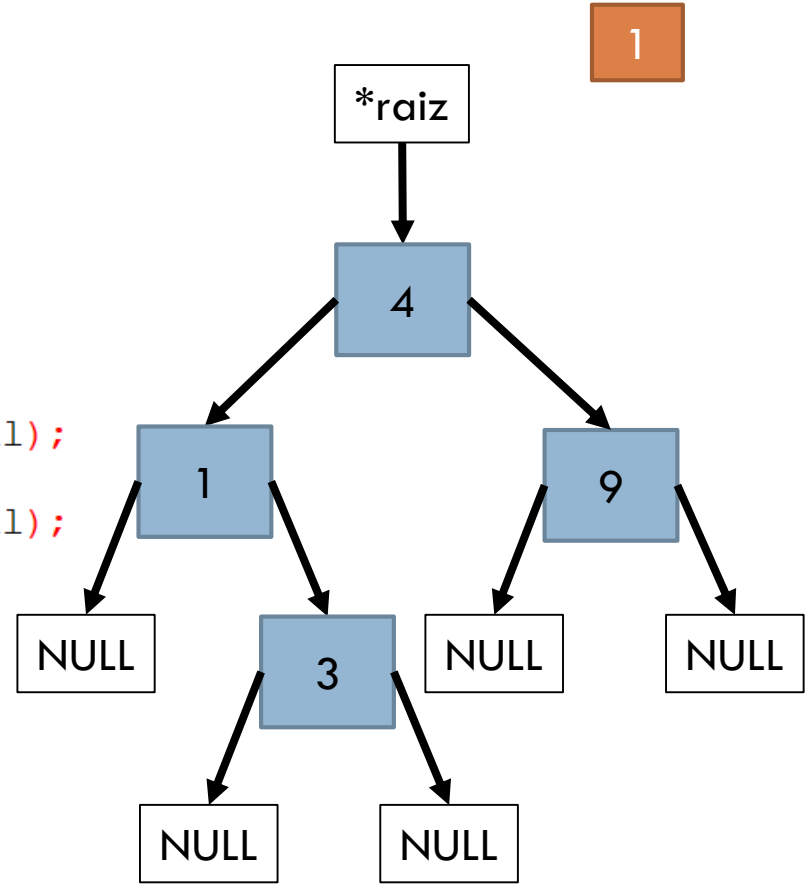
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



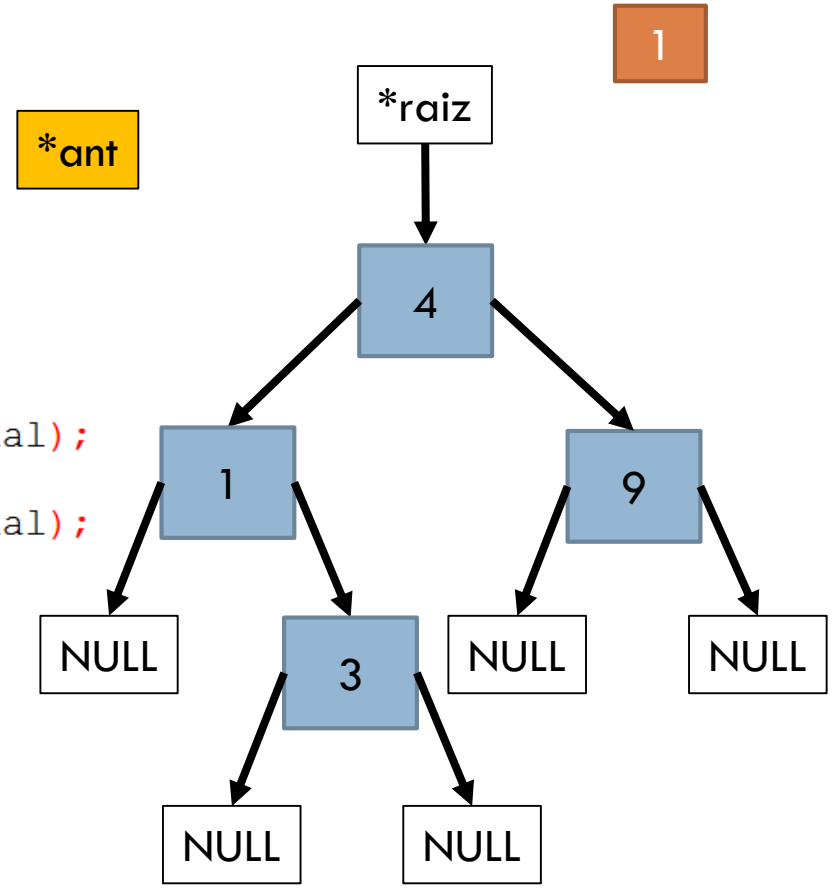
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



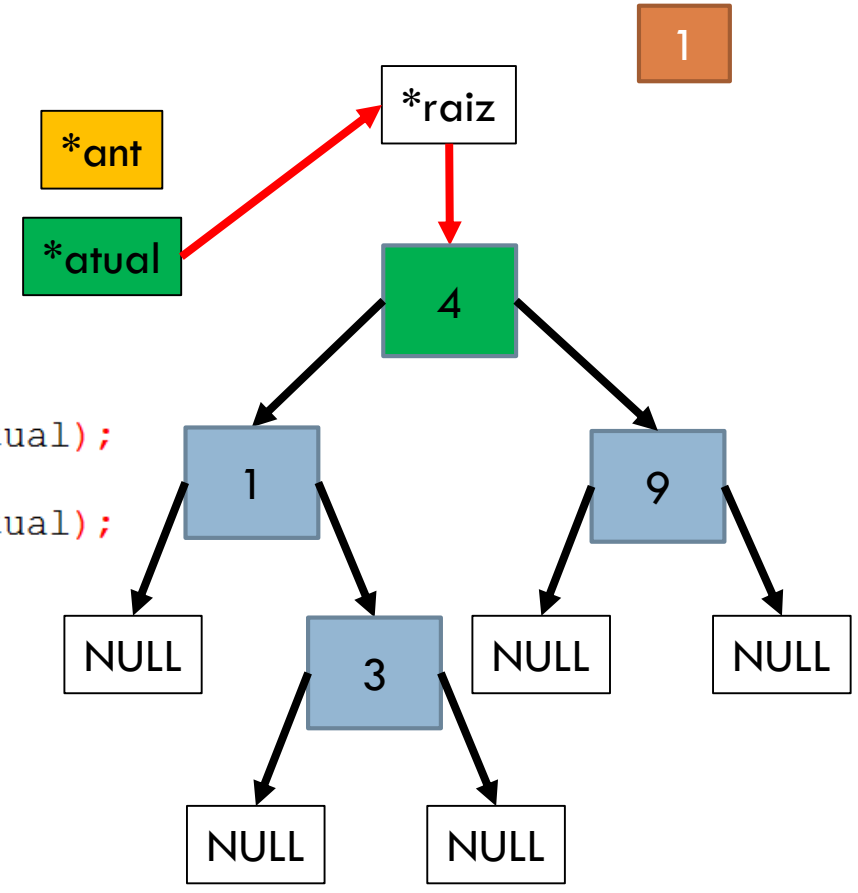
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



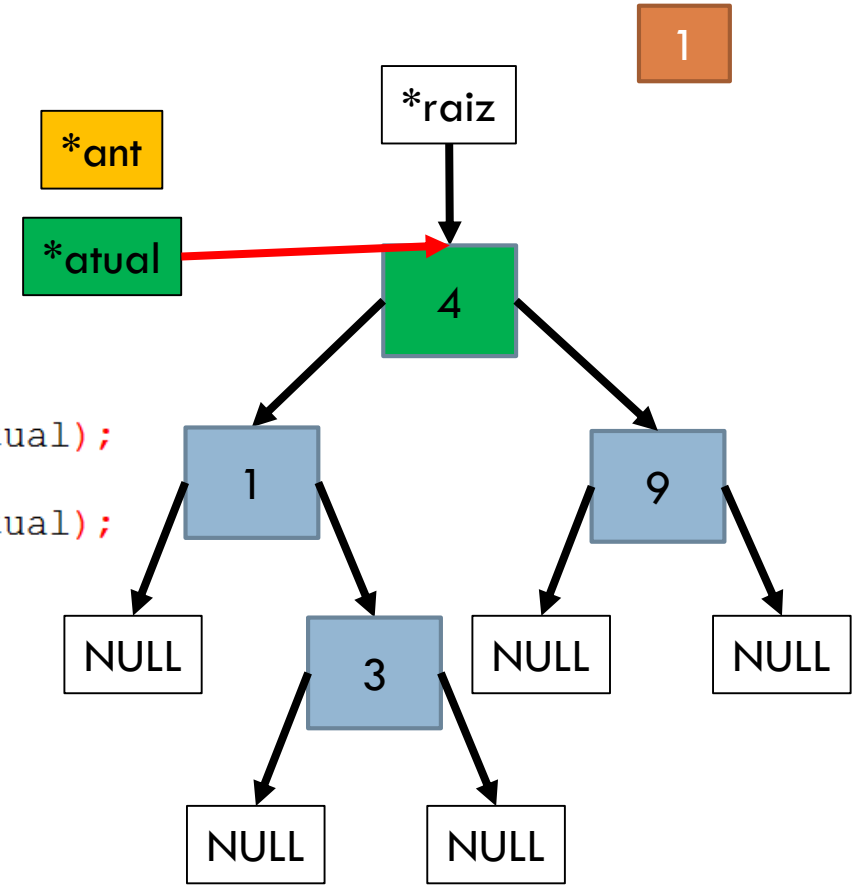
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



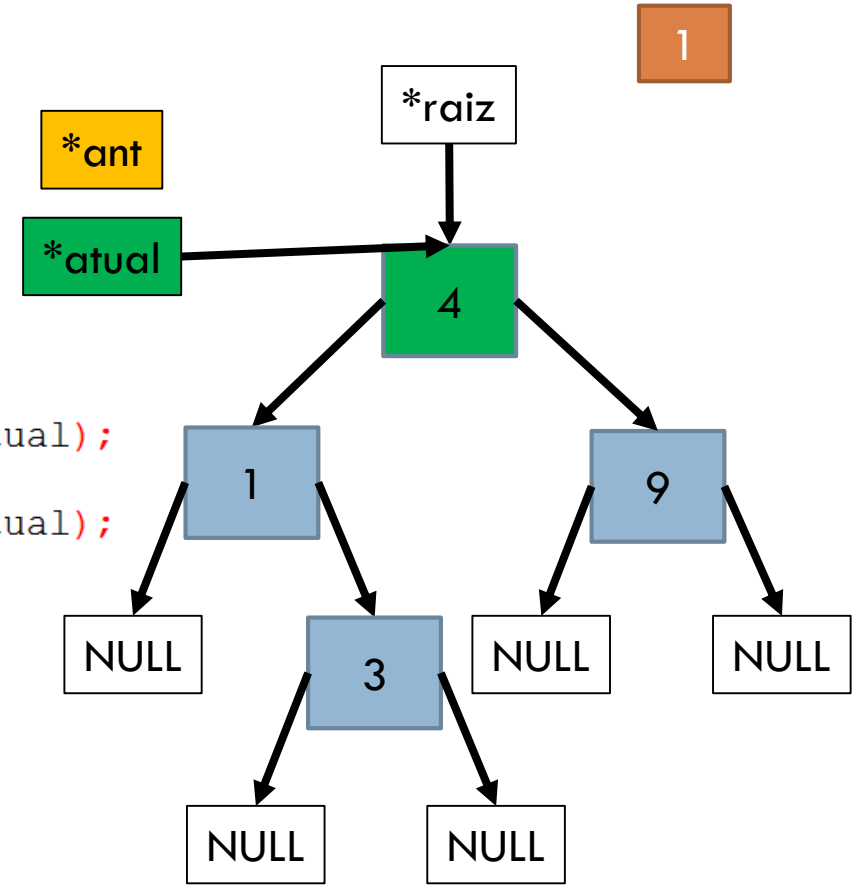
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



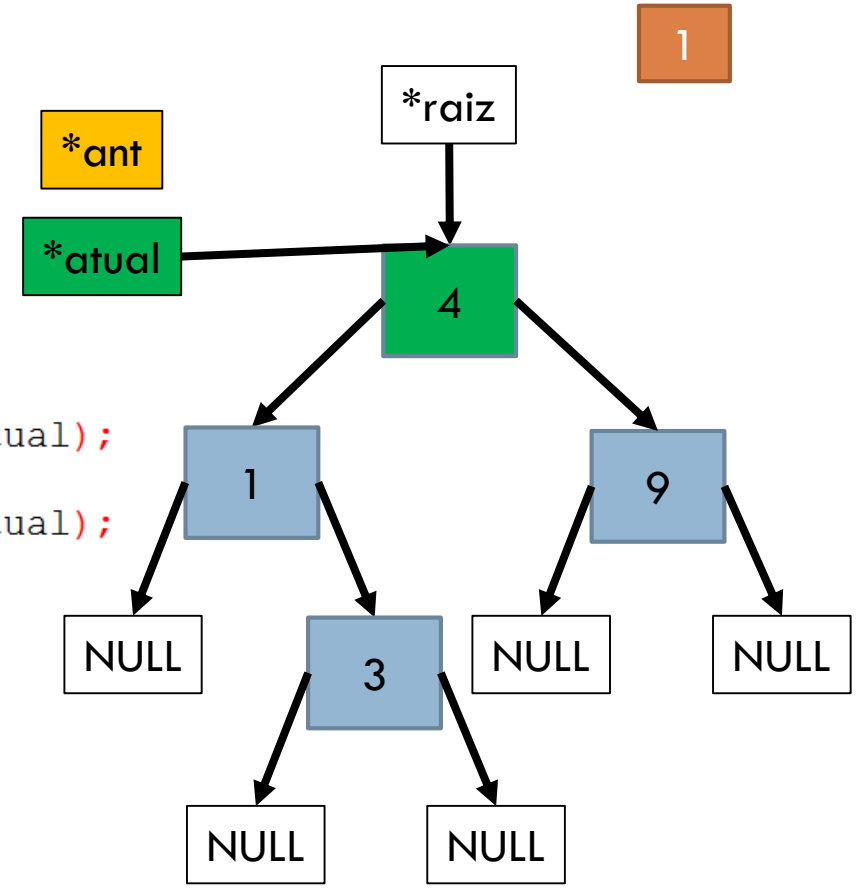
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



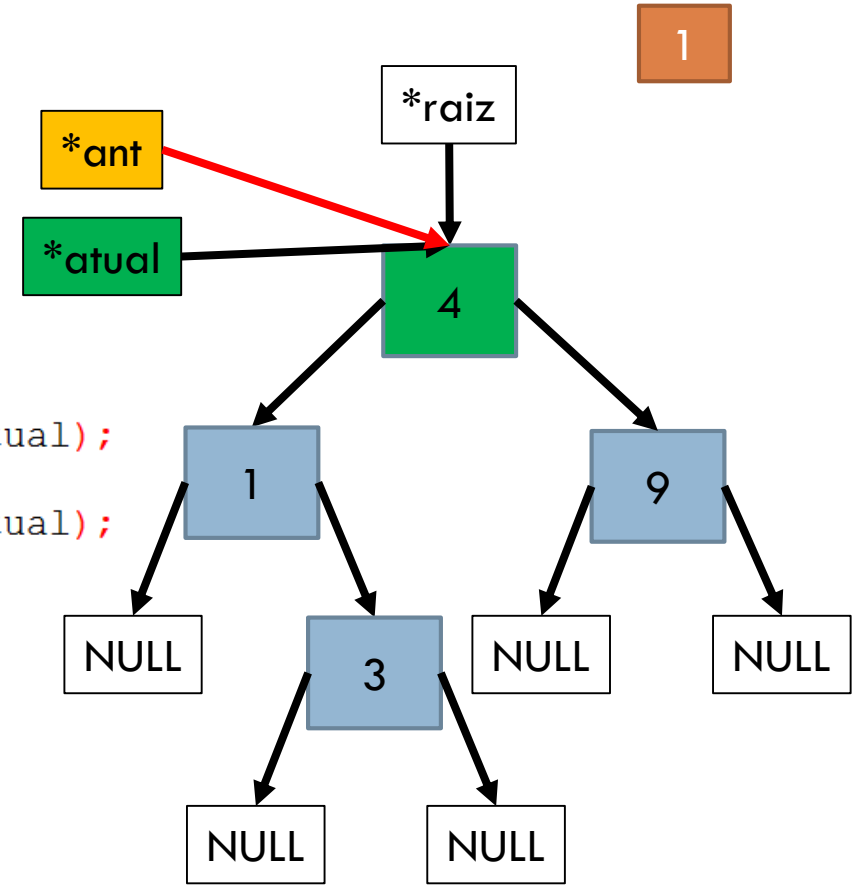
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



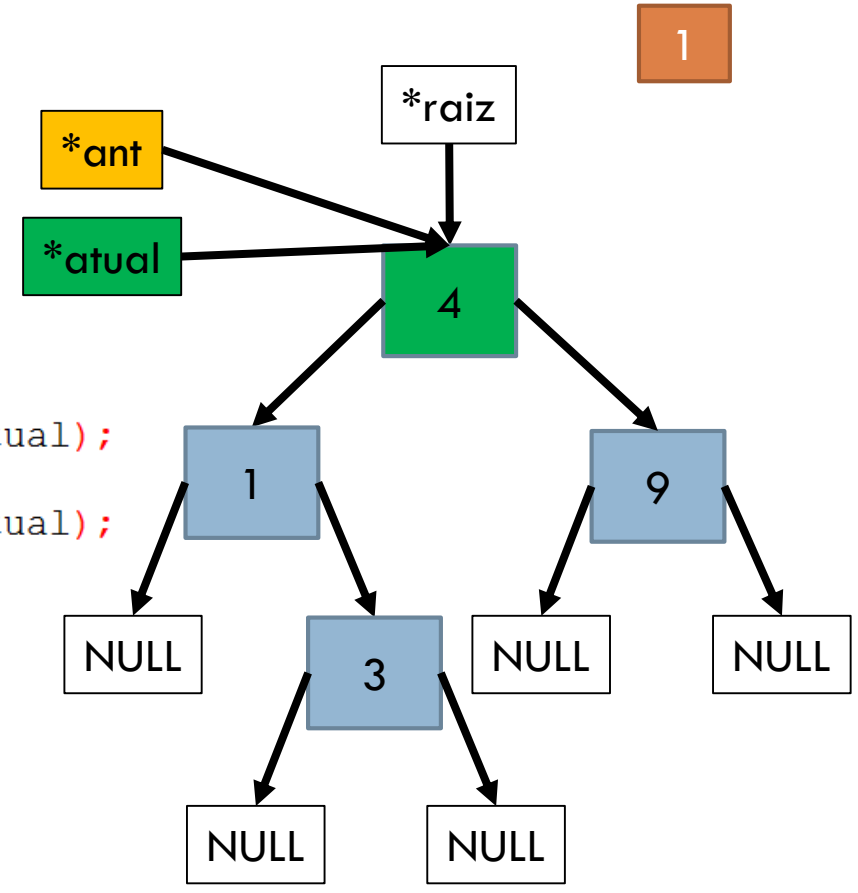
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



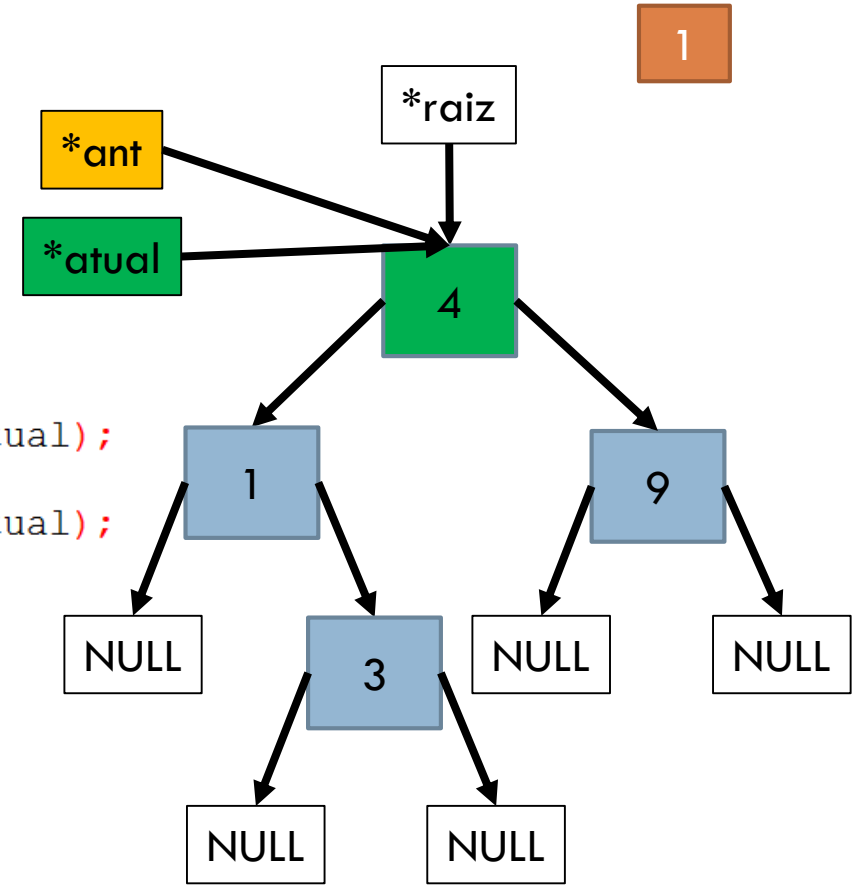
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



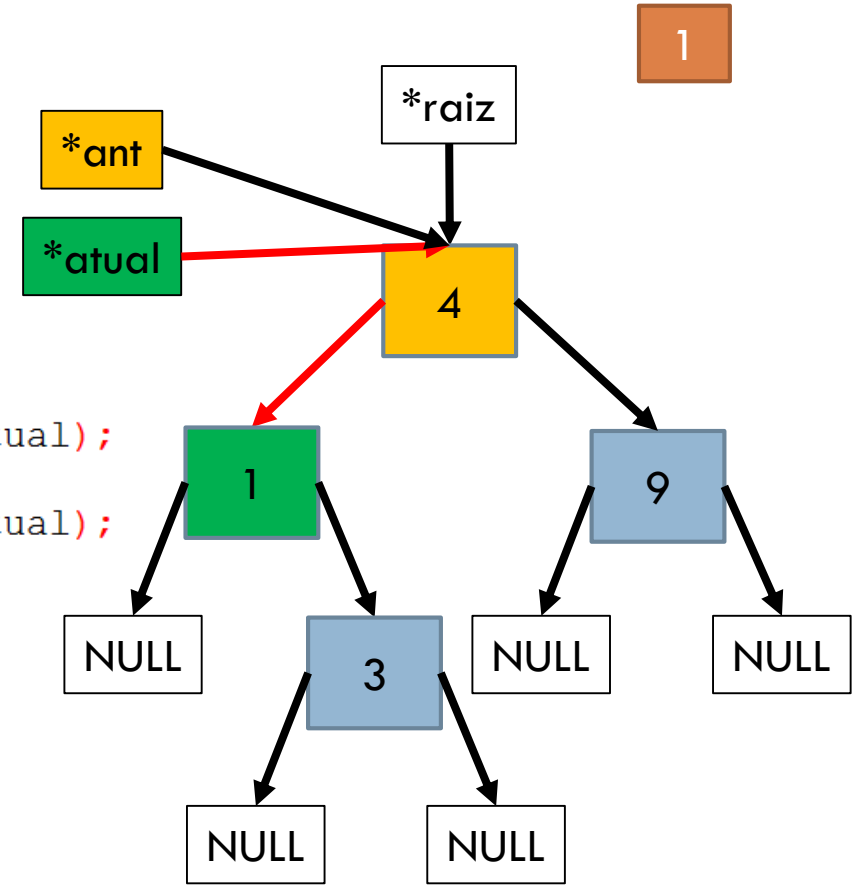
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



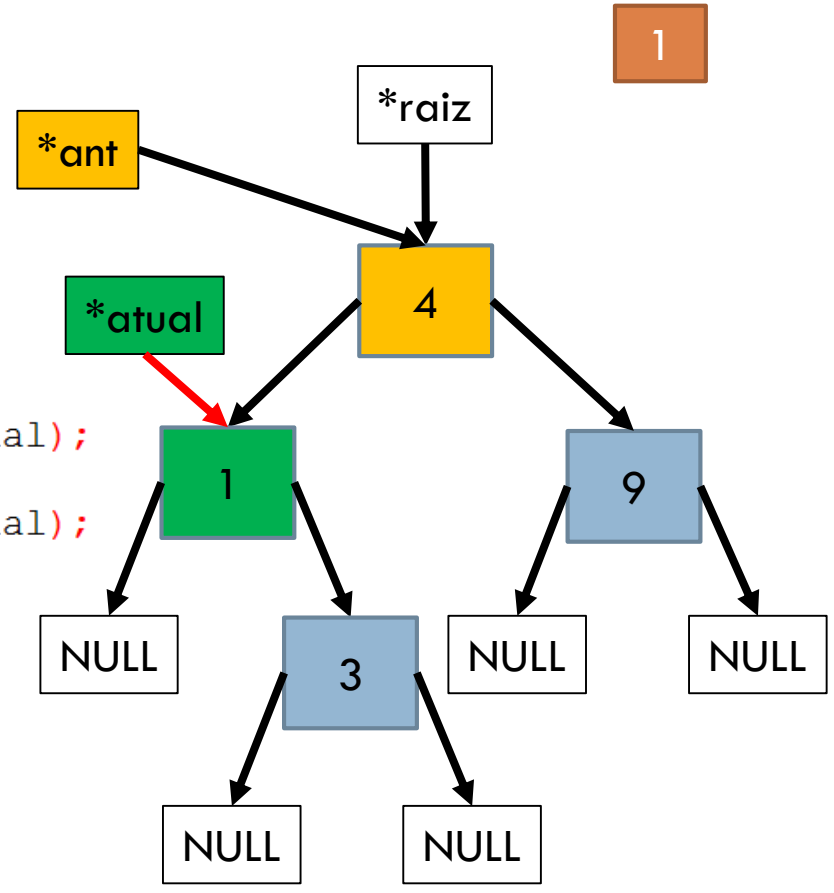
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



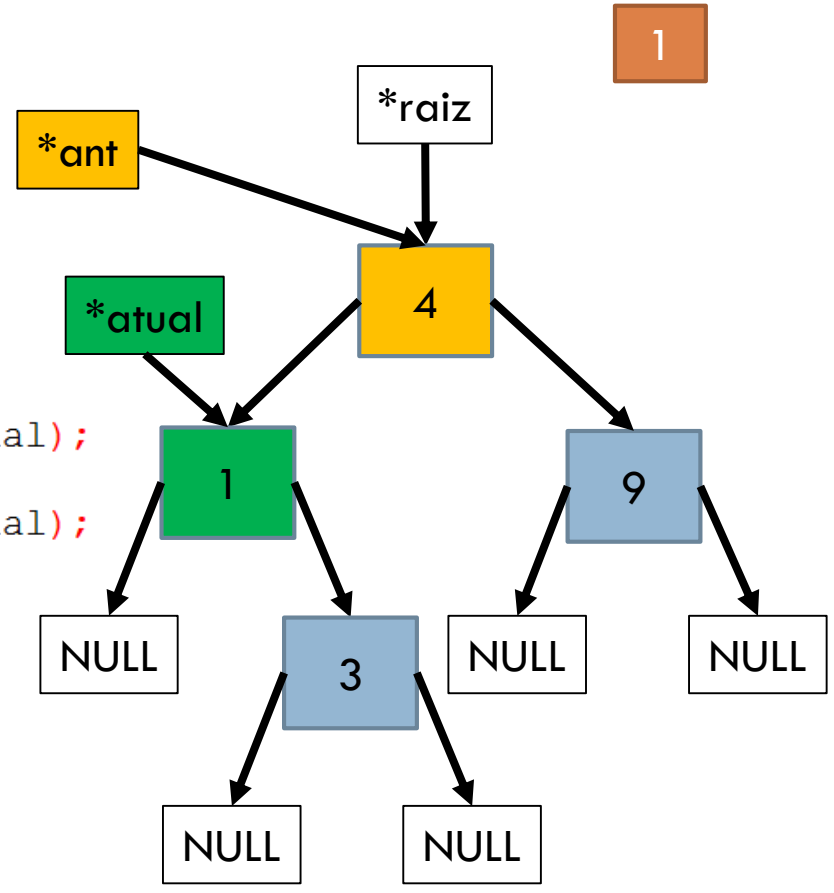
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



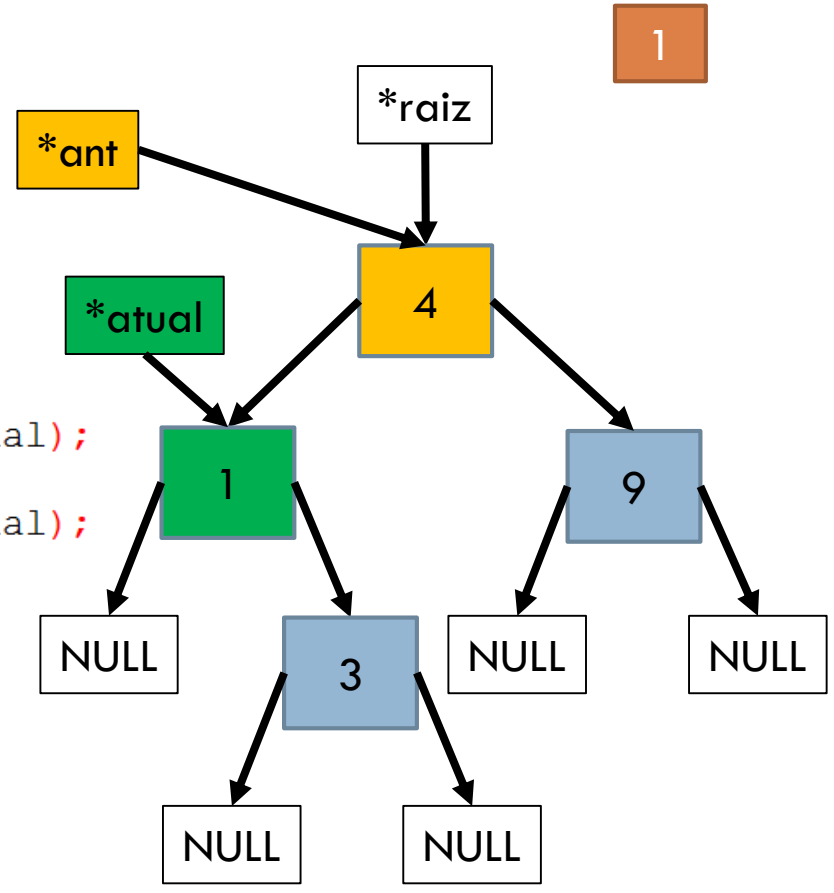
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



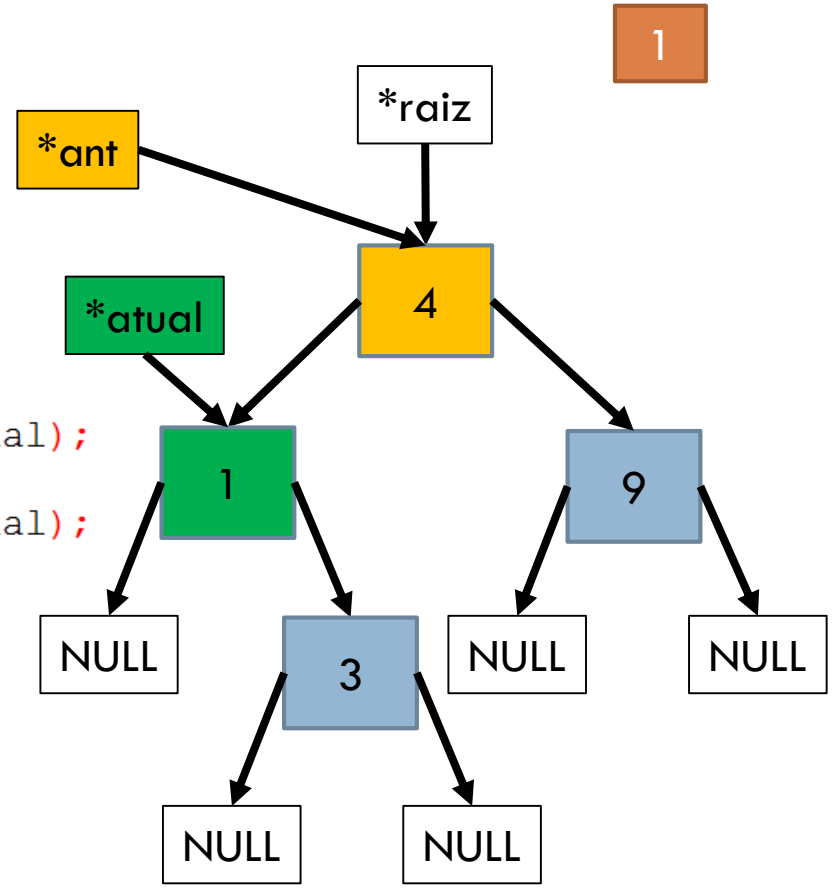
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



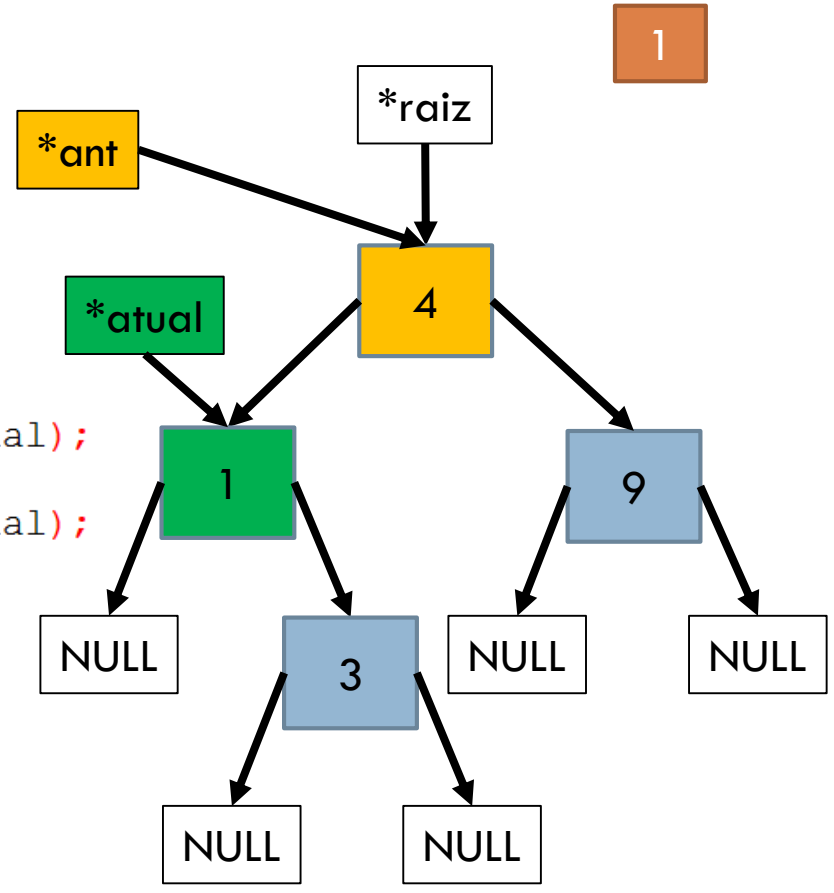
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



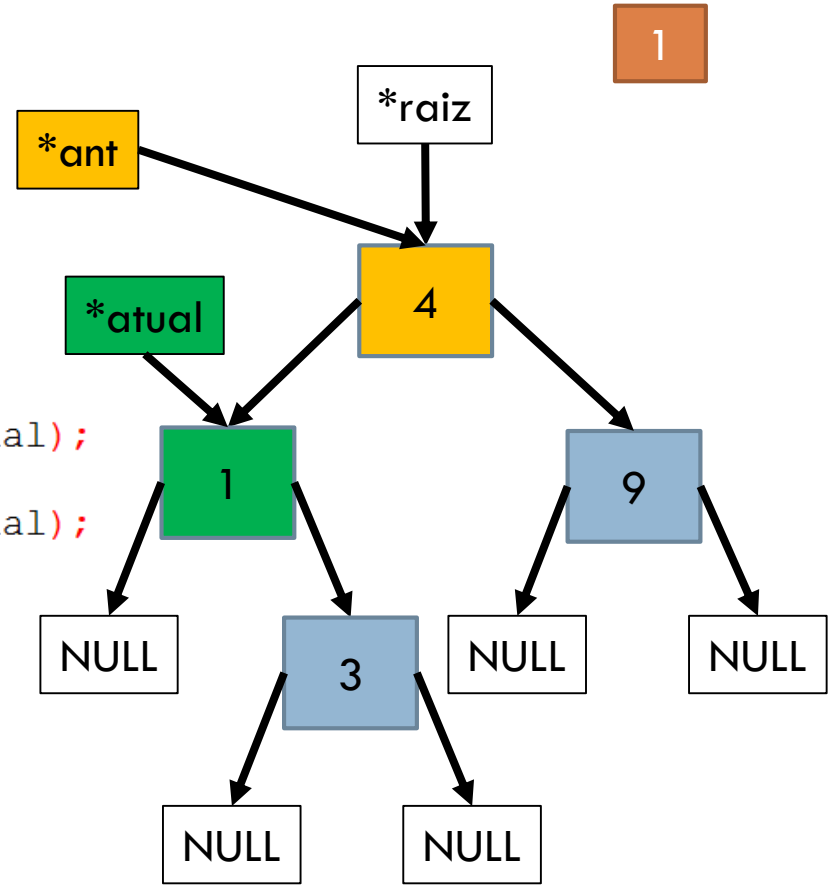
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



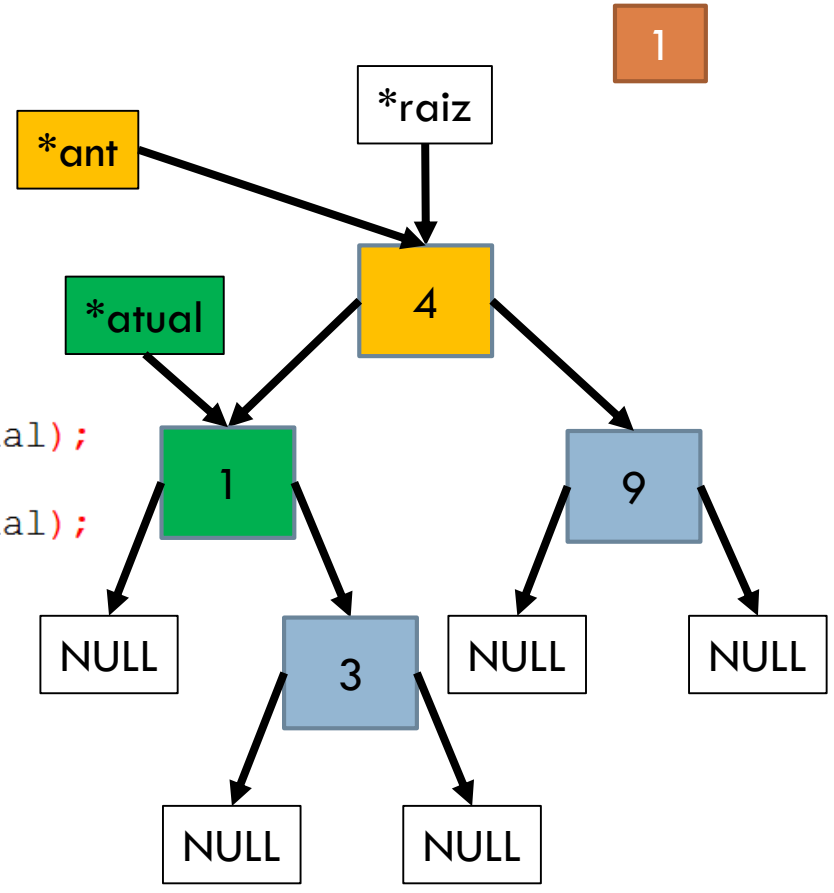
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



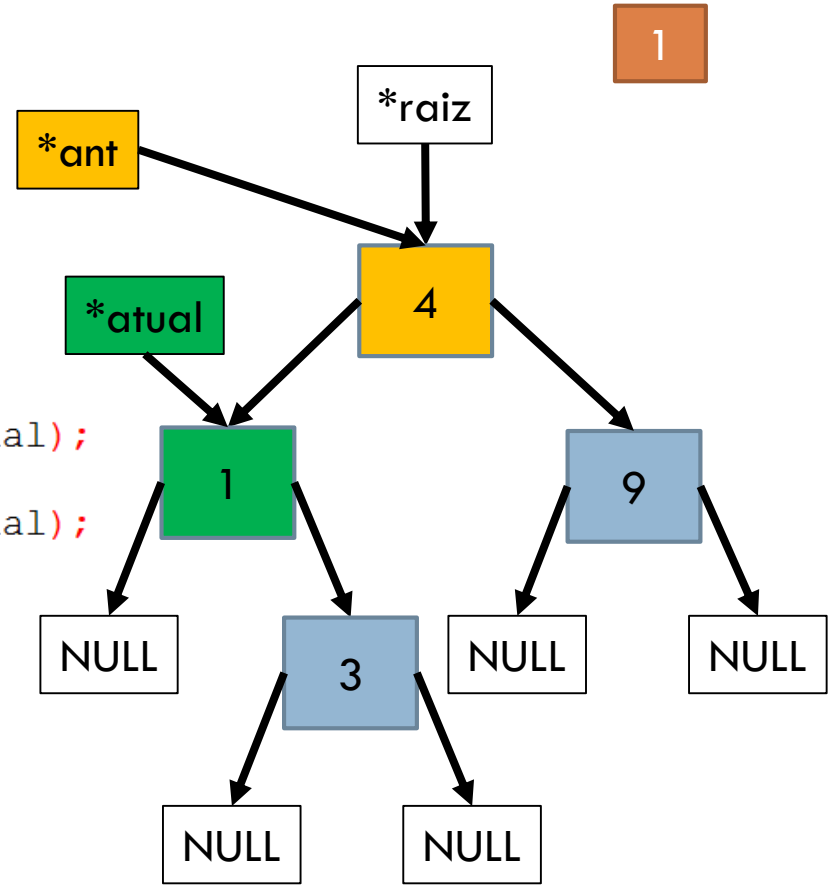
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



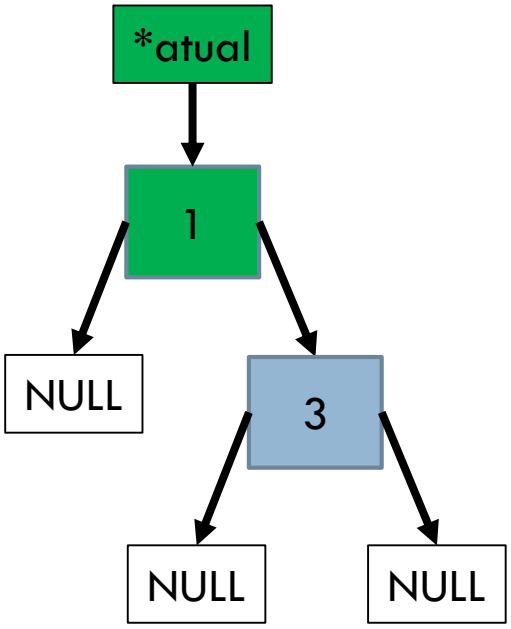
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

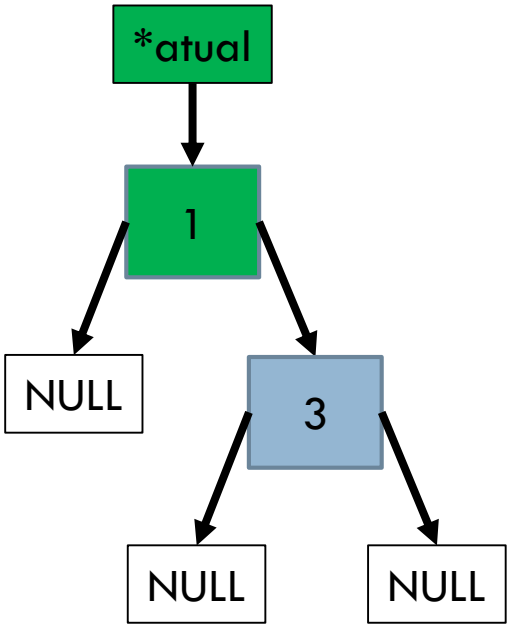


Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

*no1

*no2

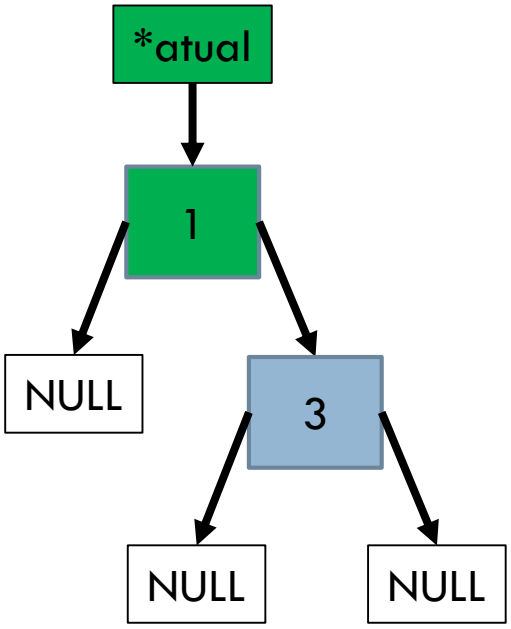


Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

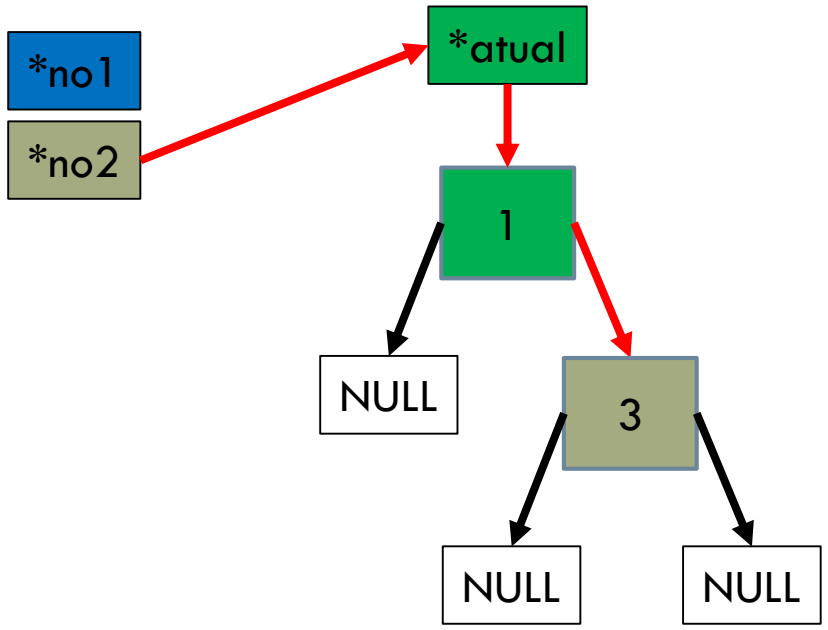
*no1

*no2



Remoção em ABB

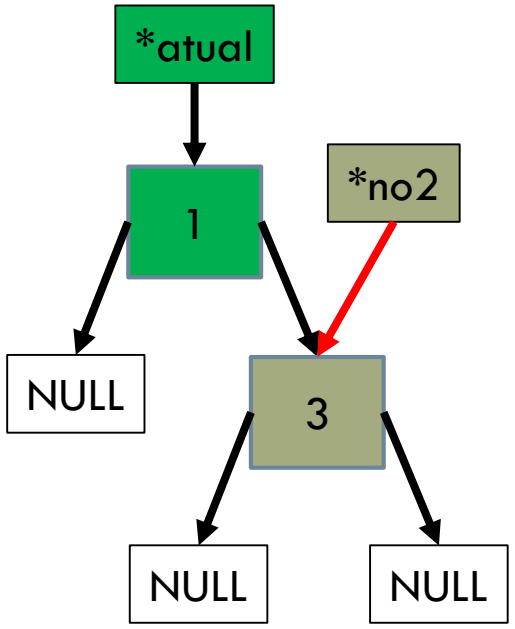

```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

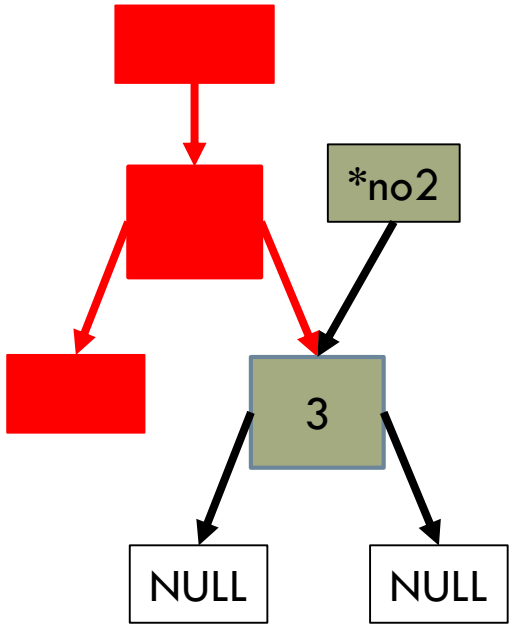
*no1



Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

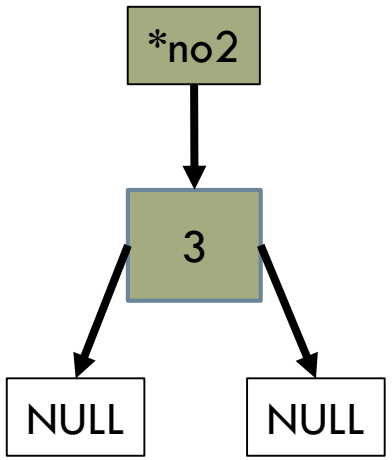
*no1



Remoção em ABB

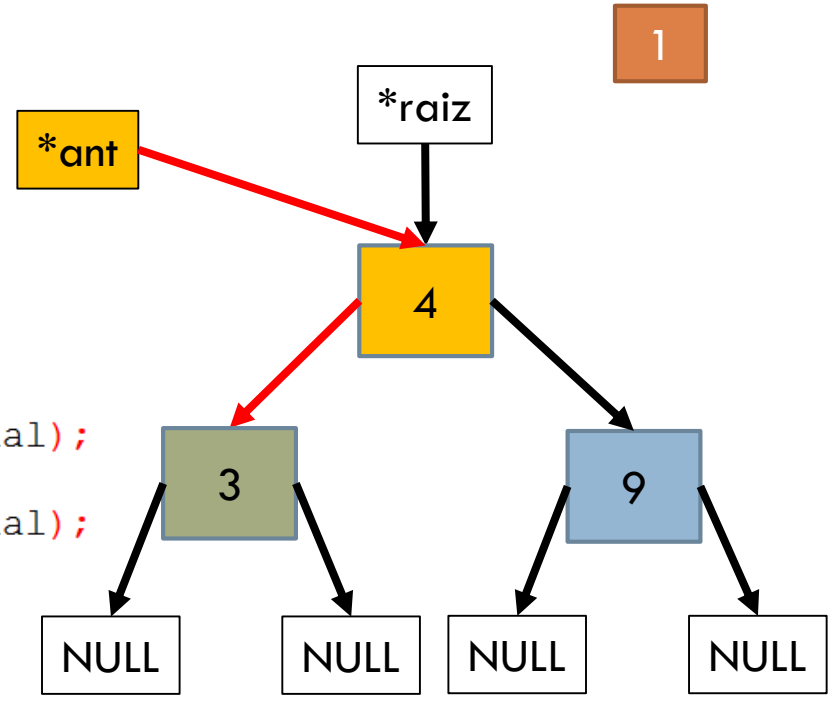

```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

*no1



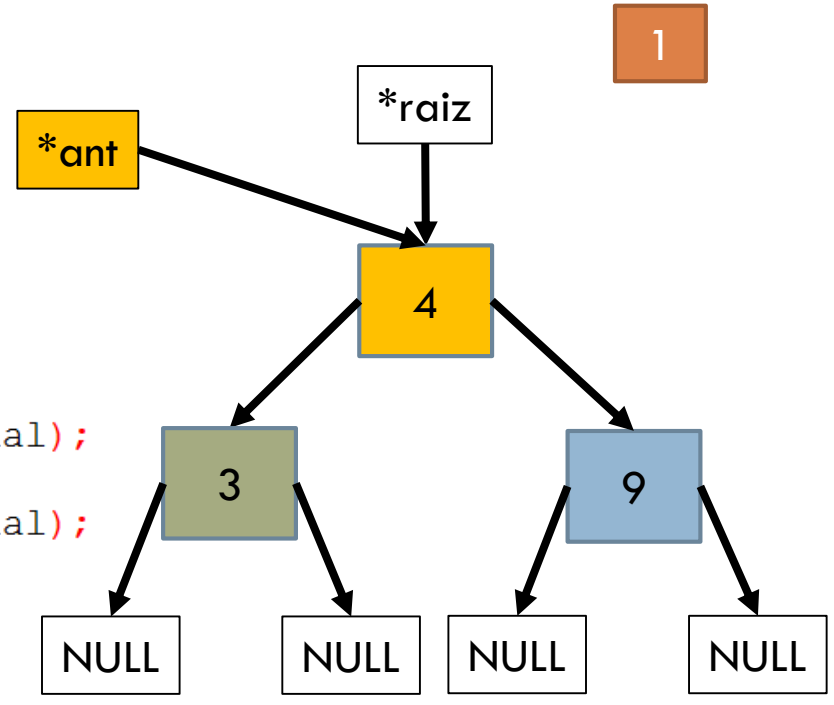
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```

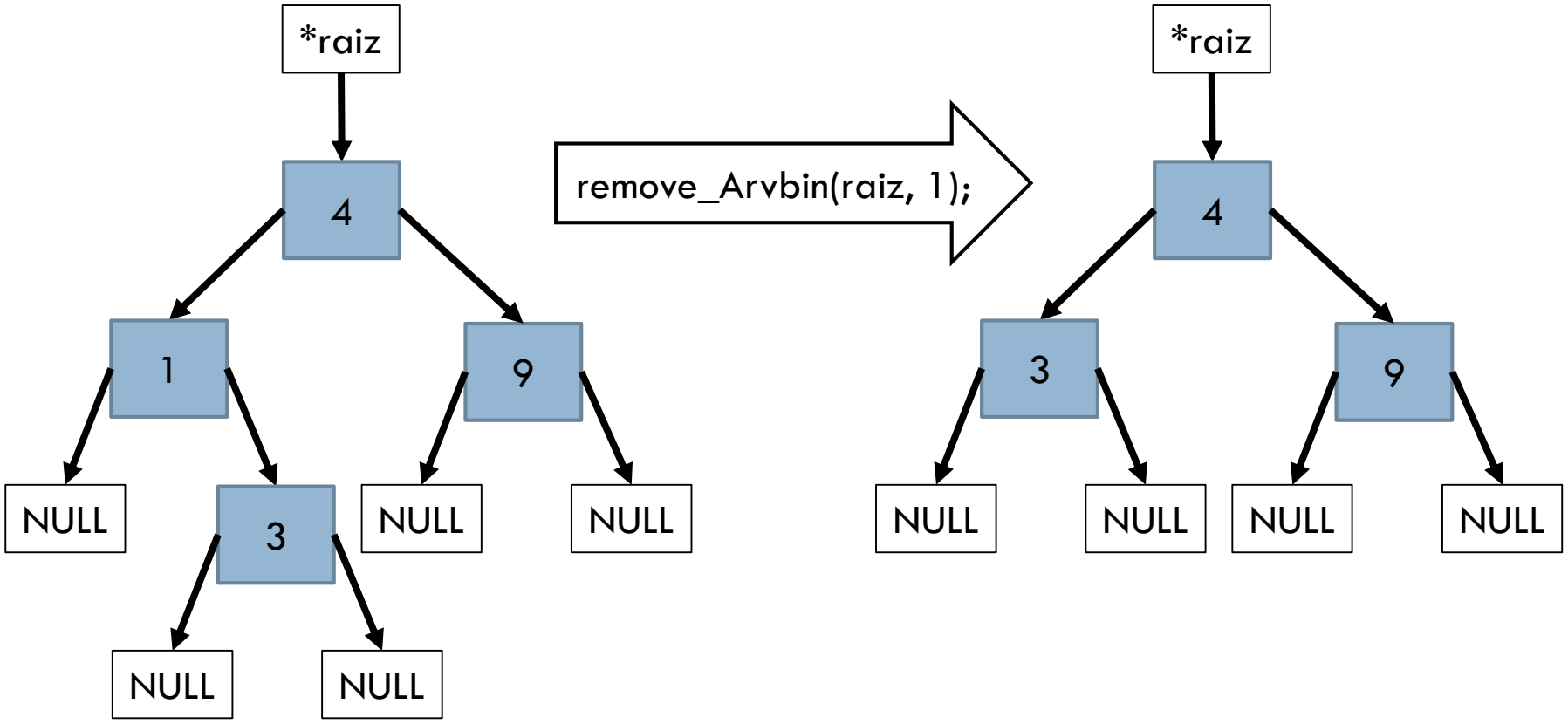


Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```

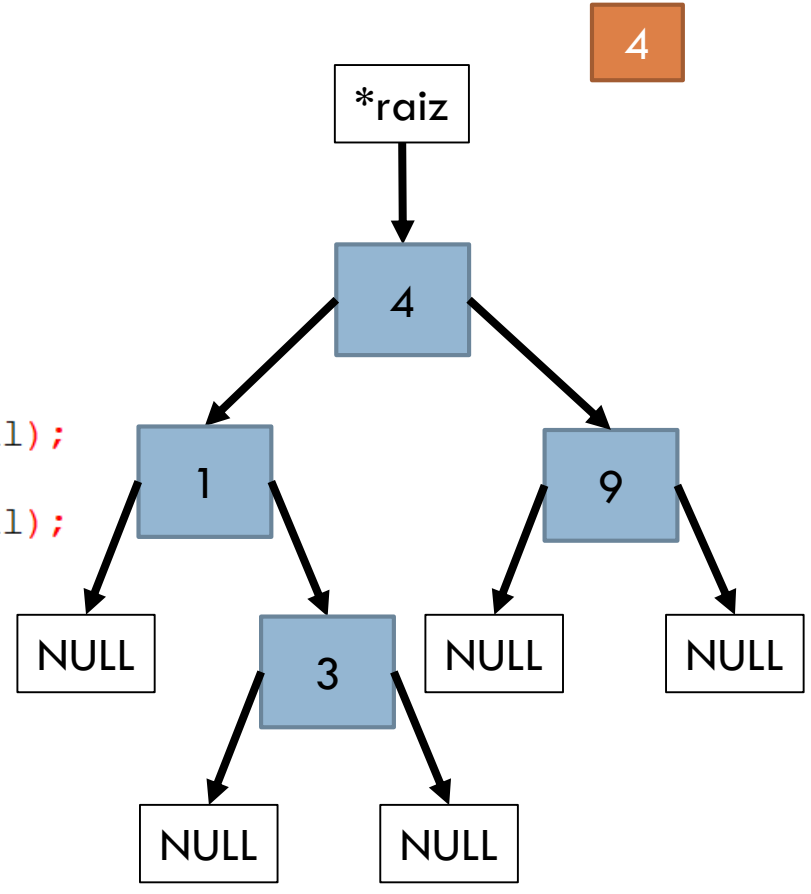


Remoção em ABB



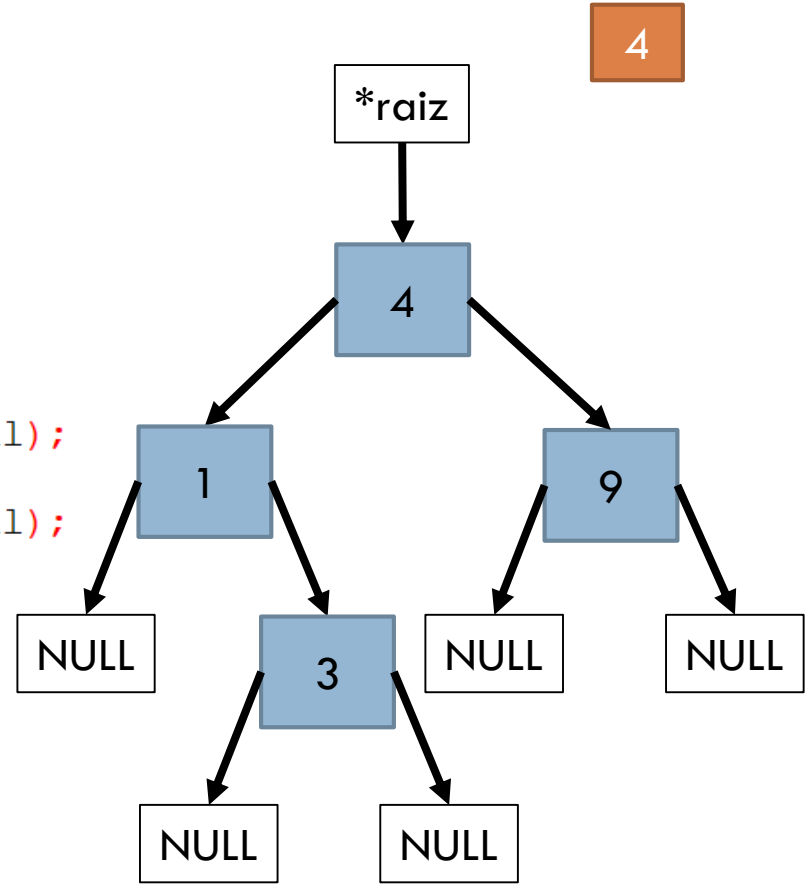
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



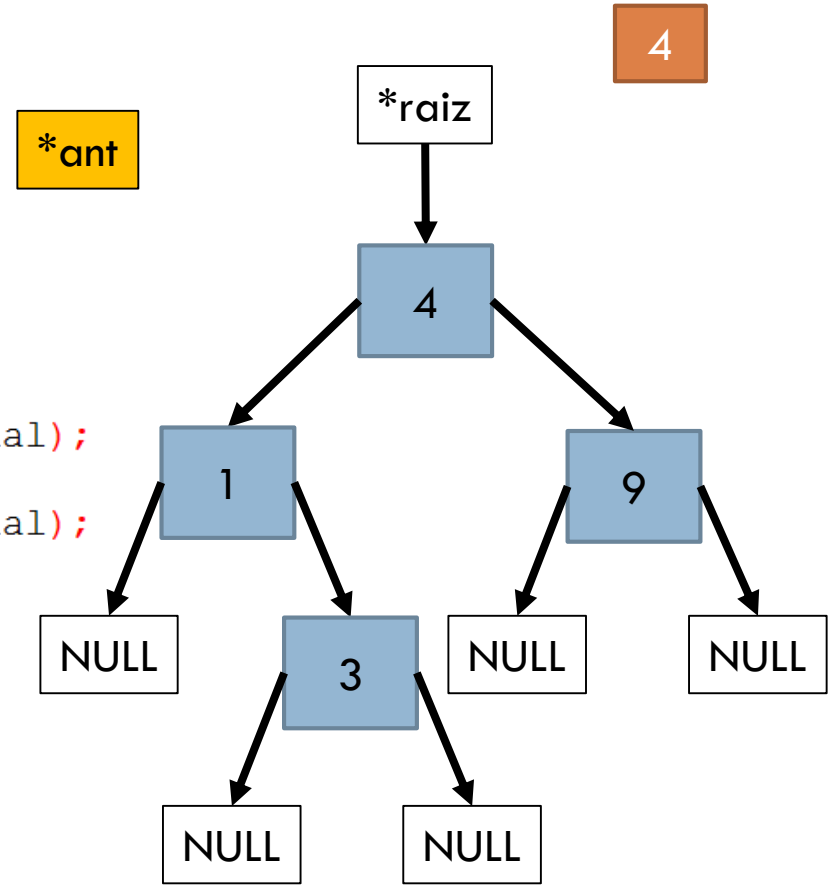
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



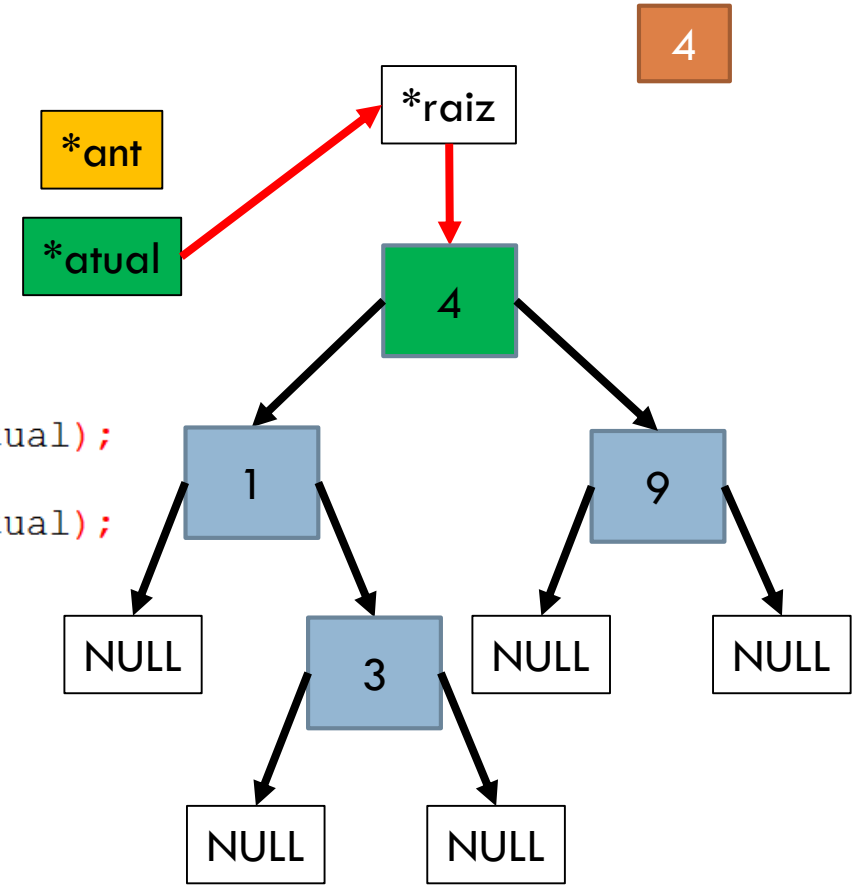
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



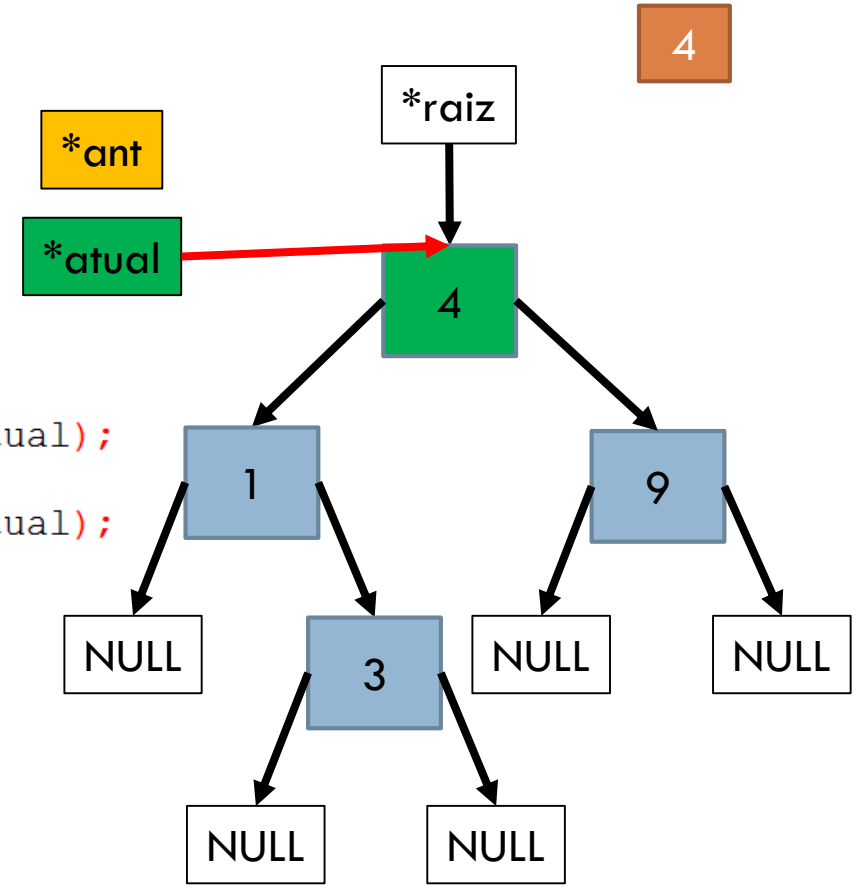
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



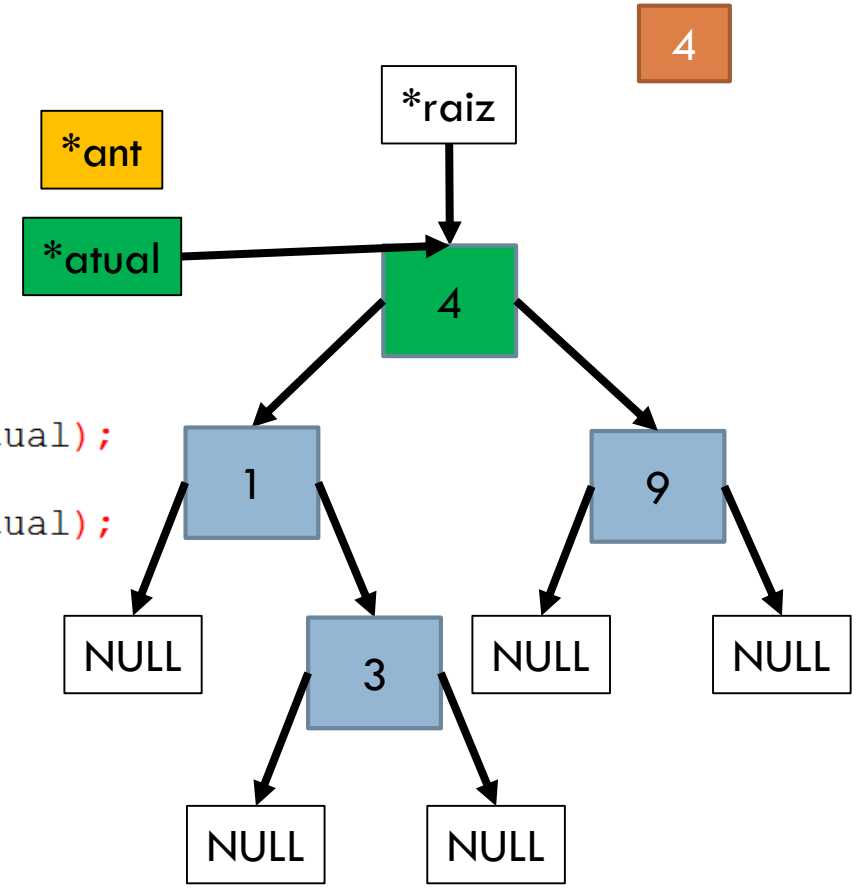
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



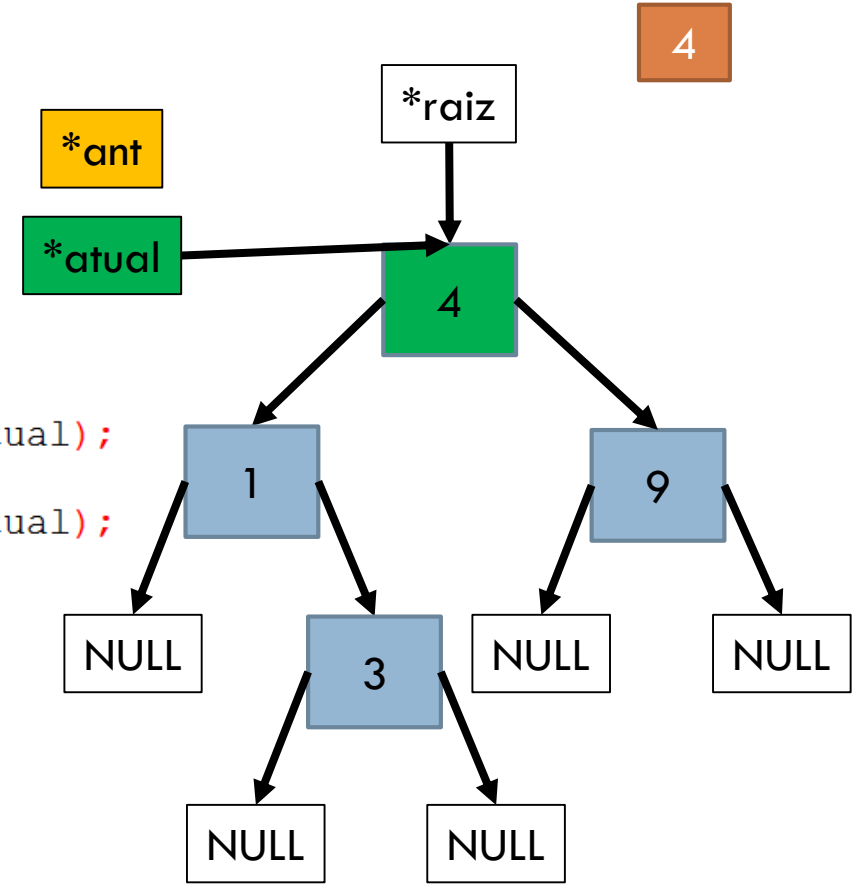
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



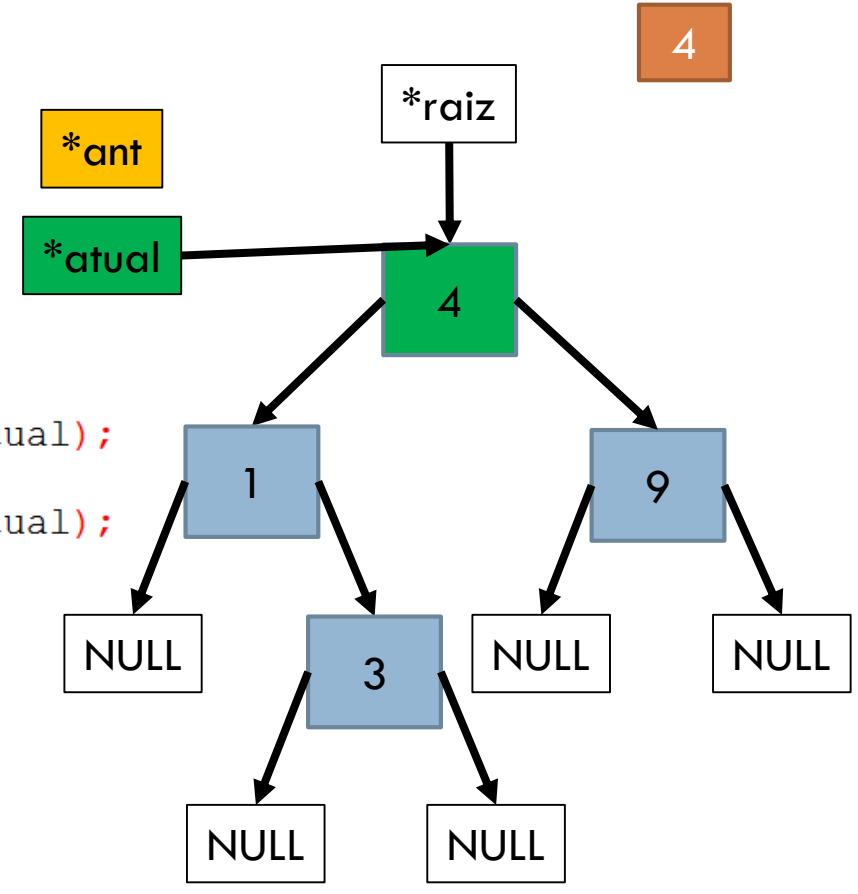
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



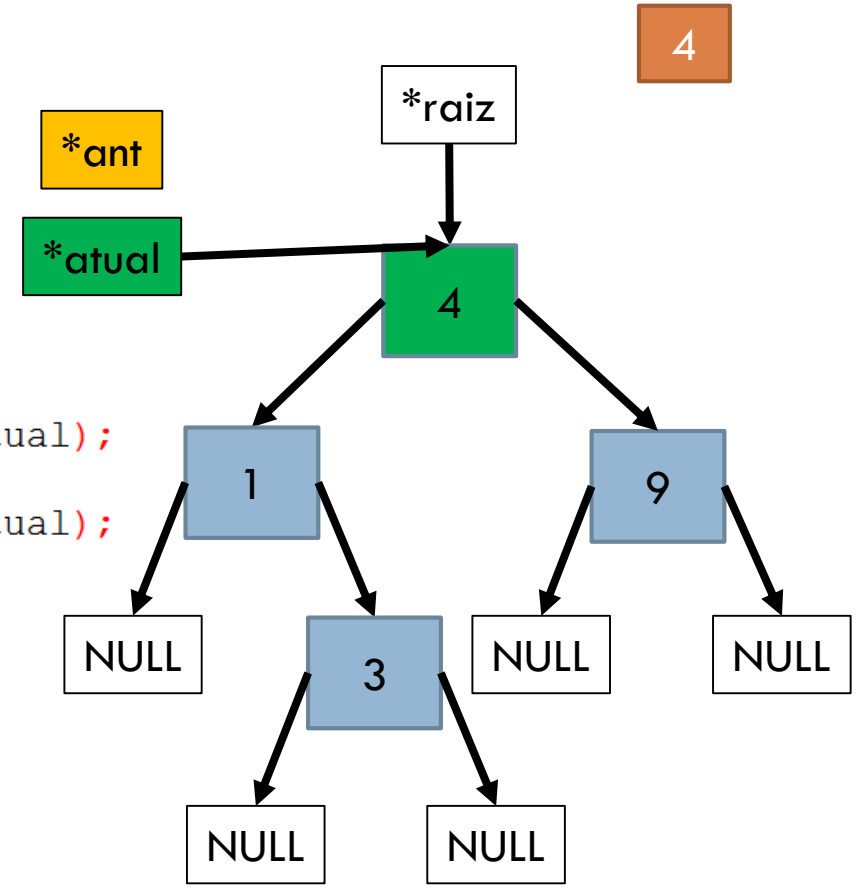
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



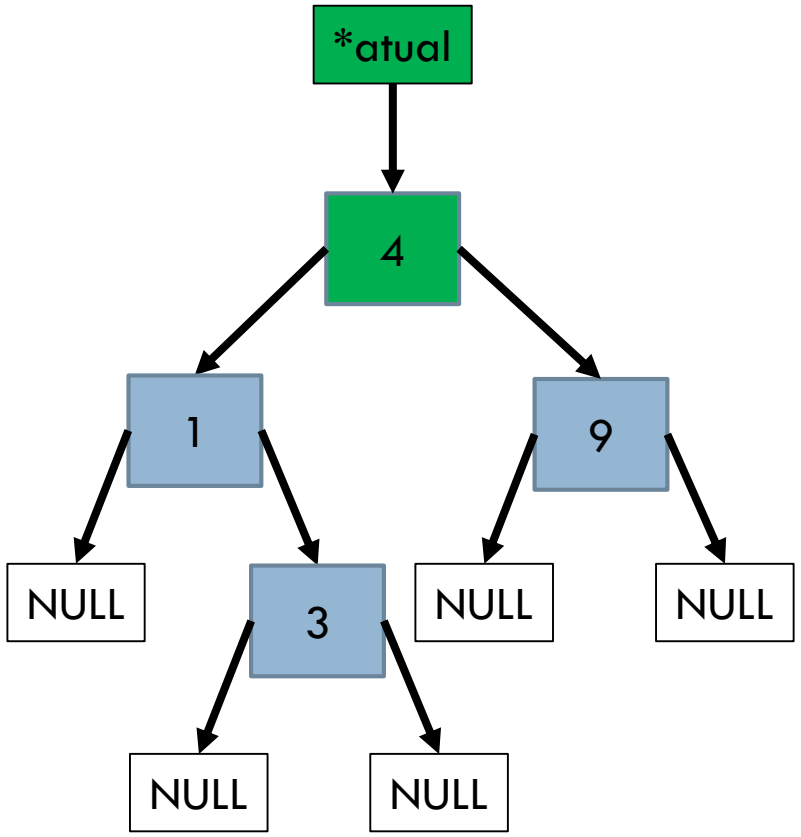
Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



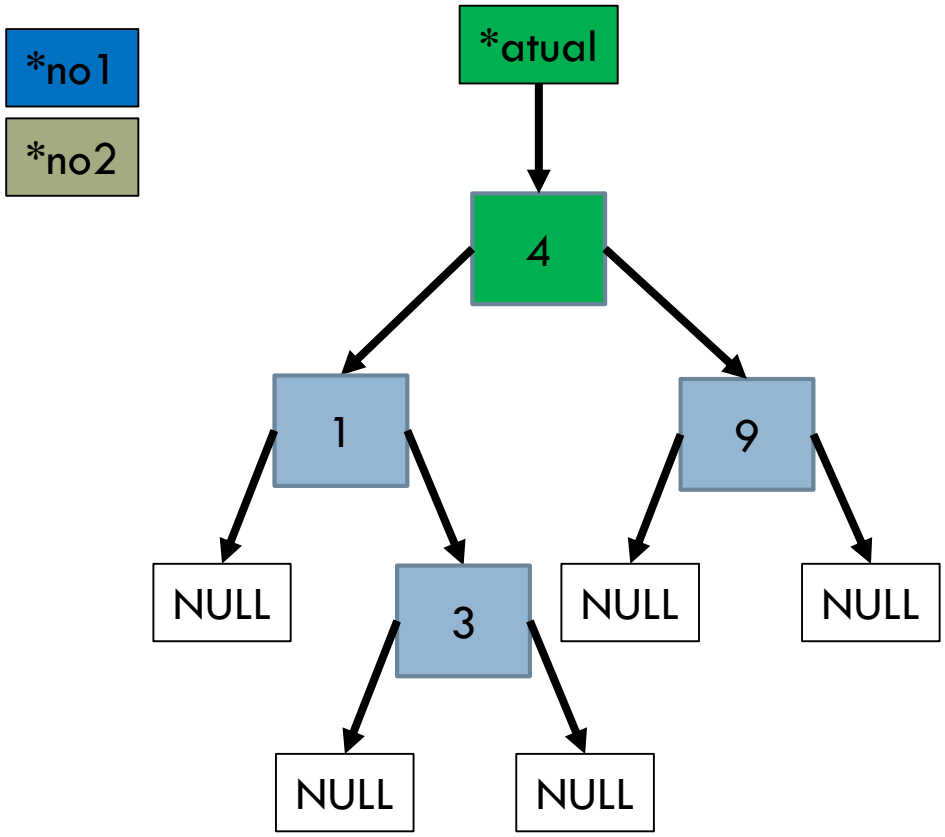
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



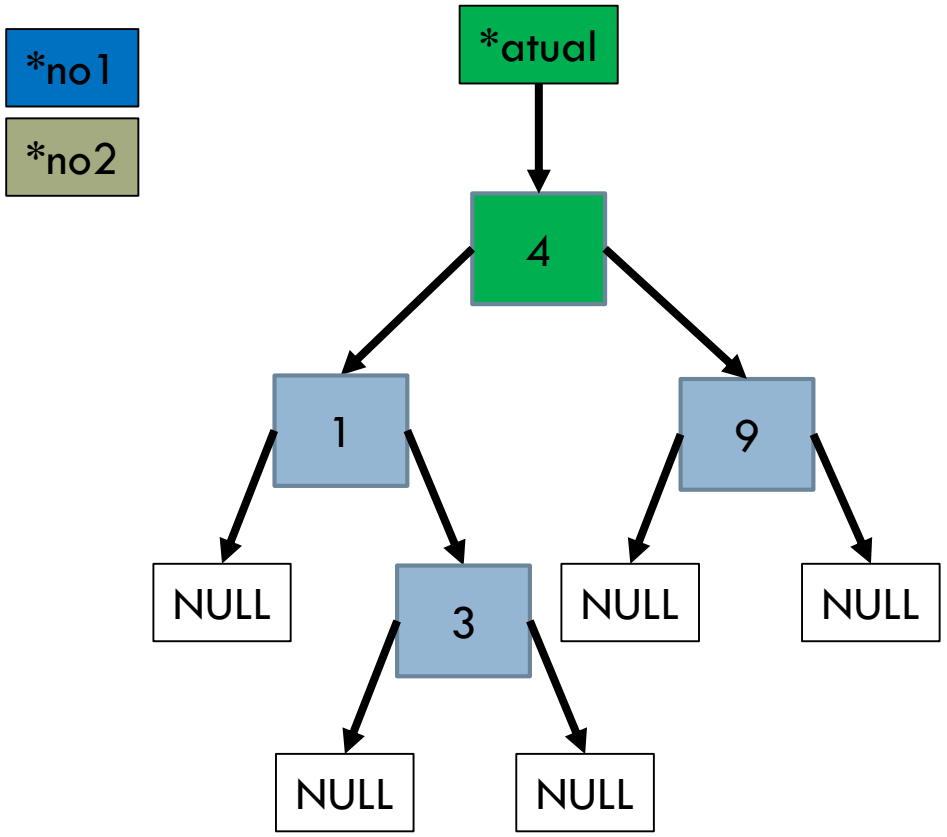
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



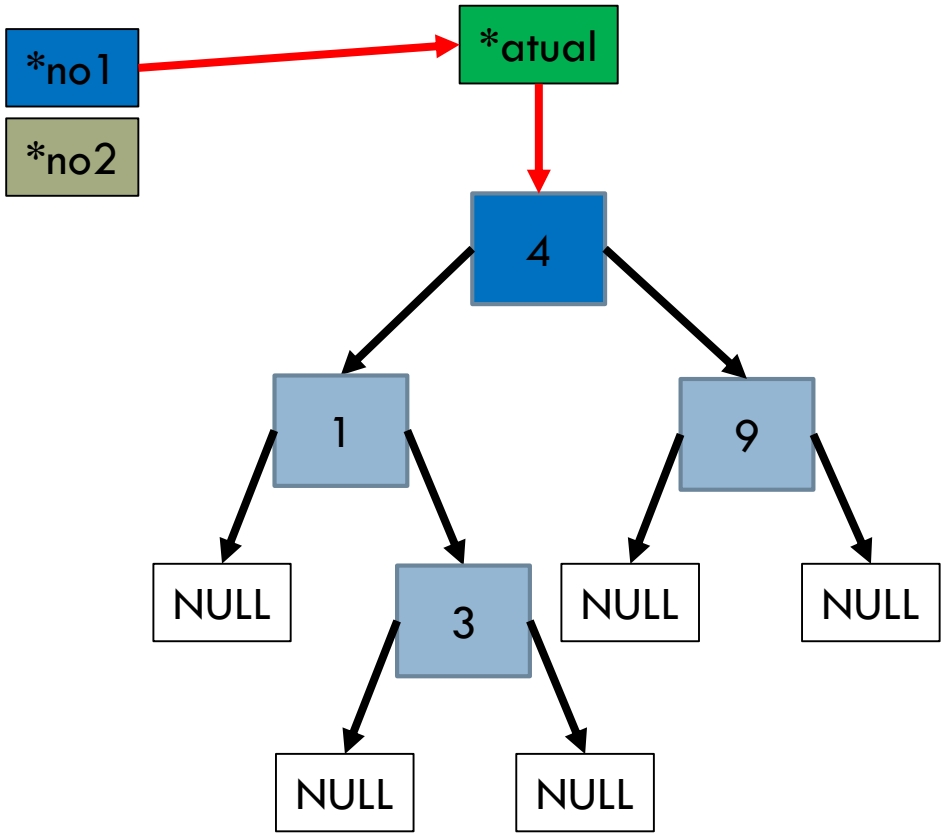
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



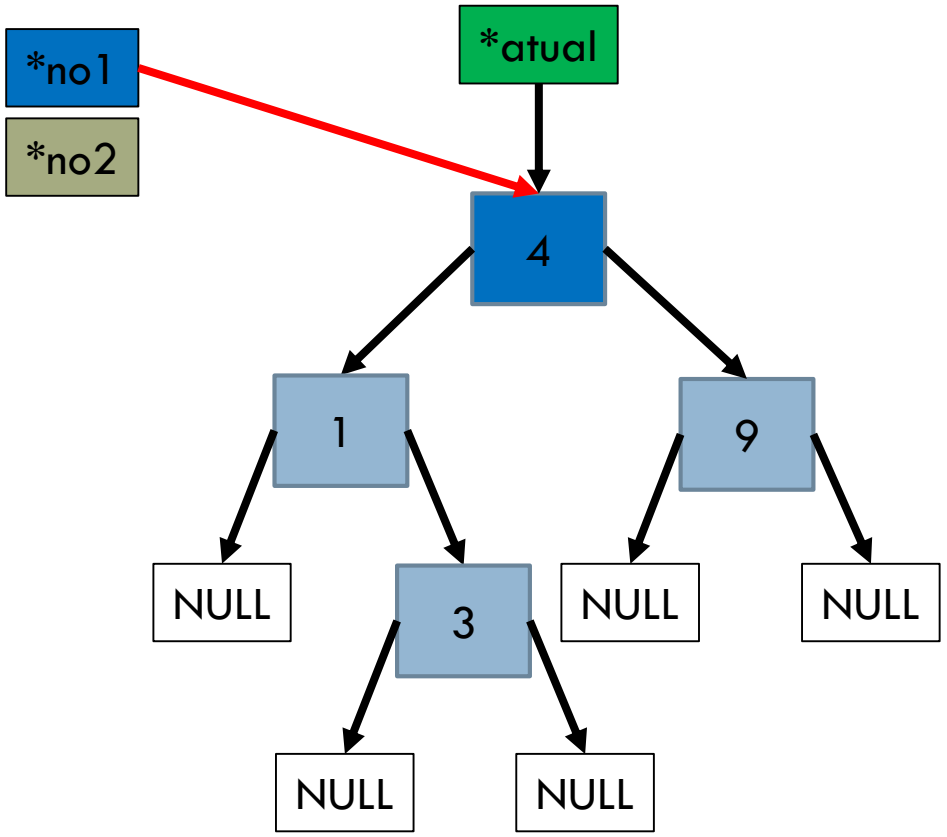
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



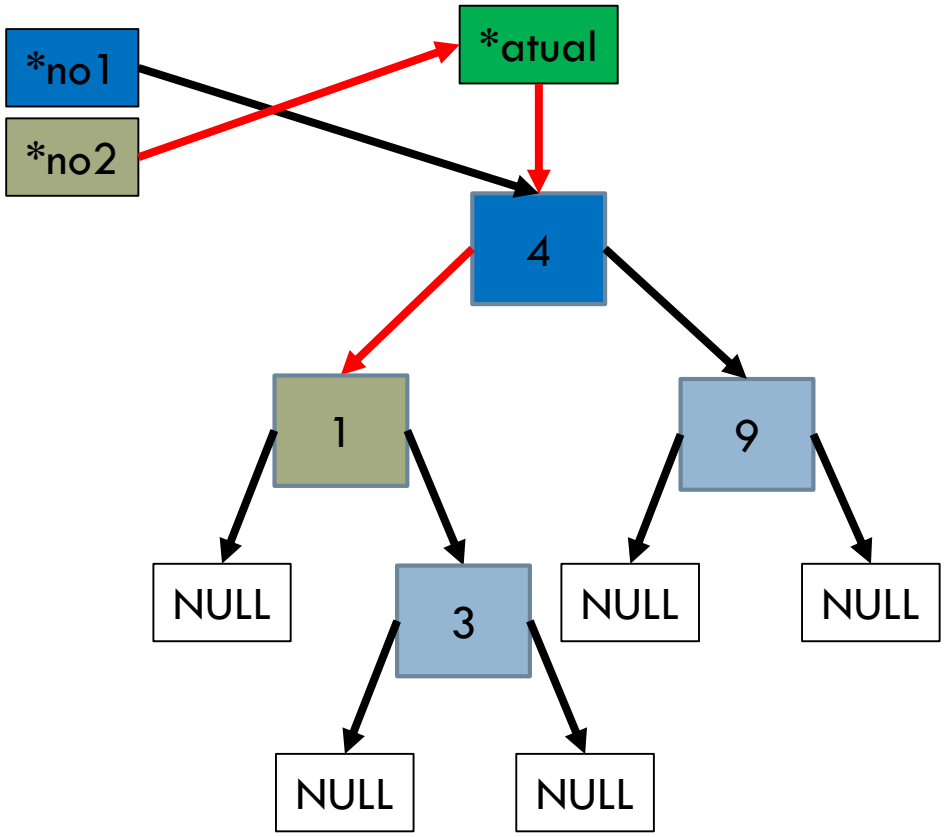
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



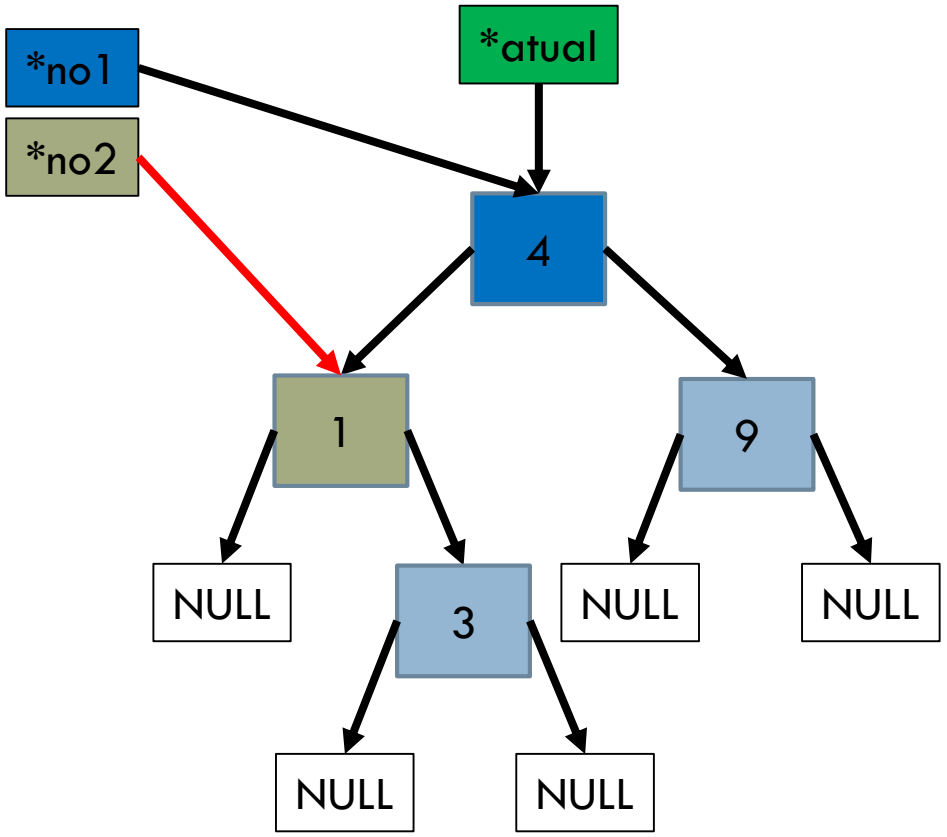
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



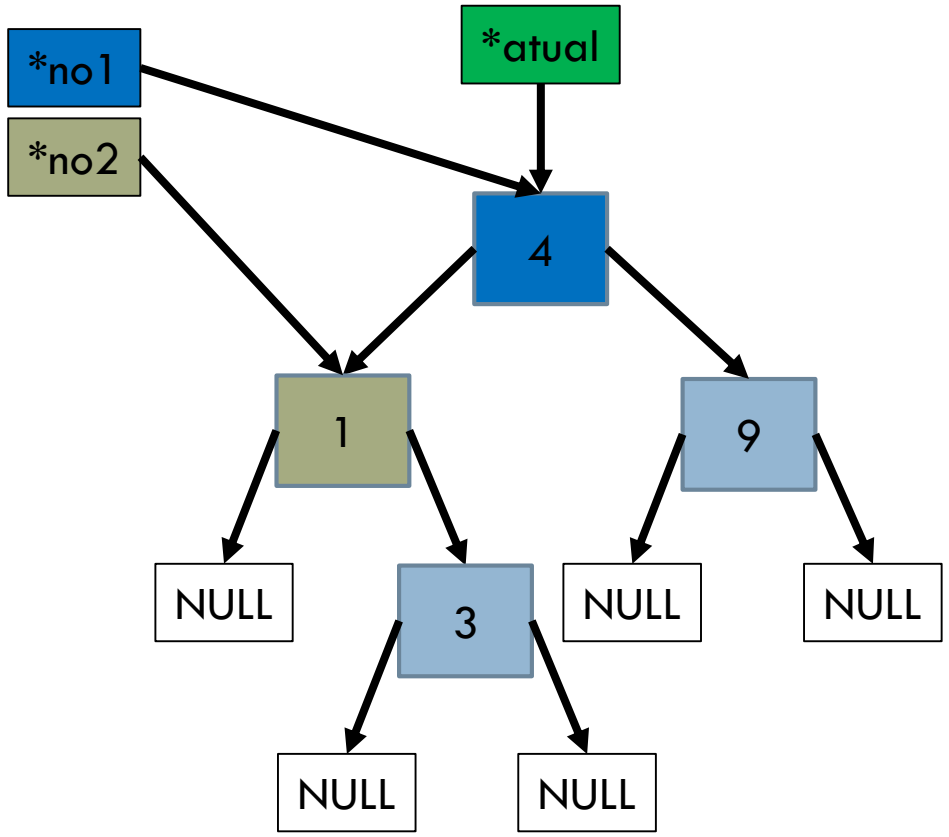
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



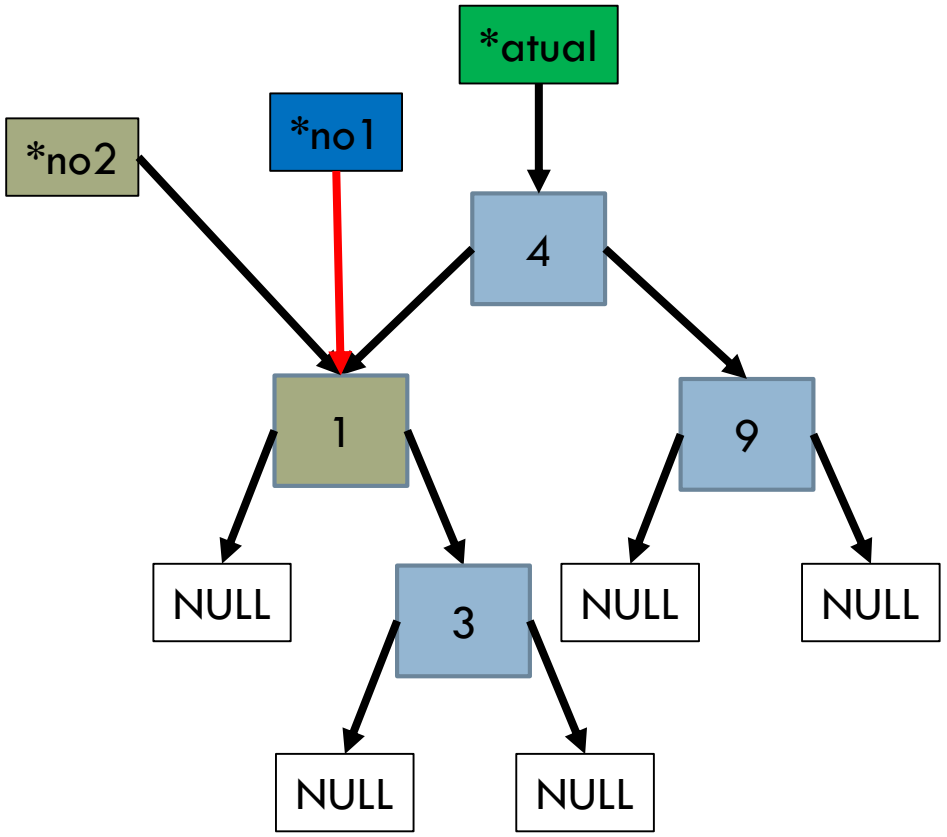
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



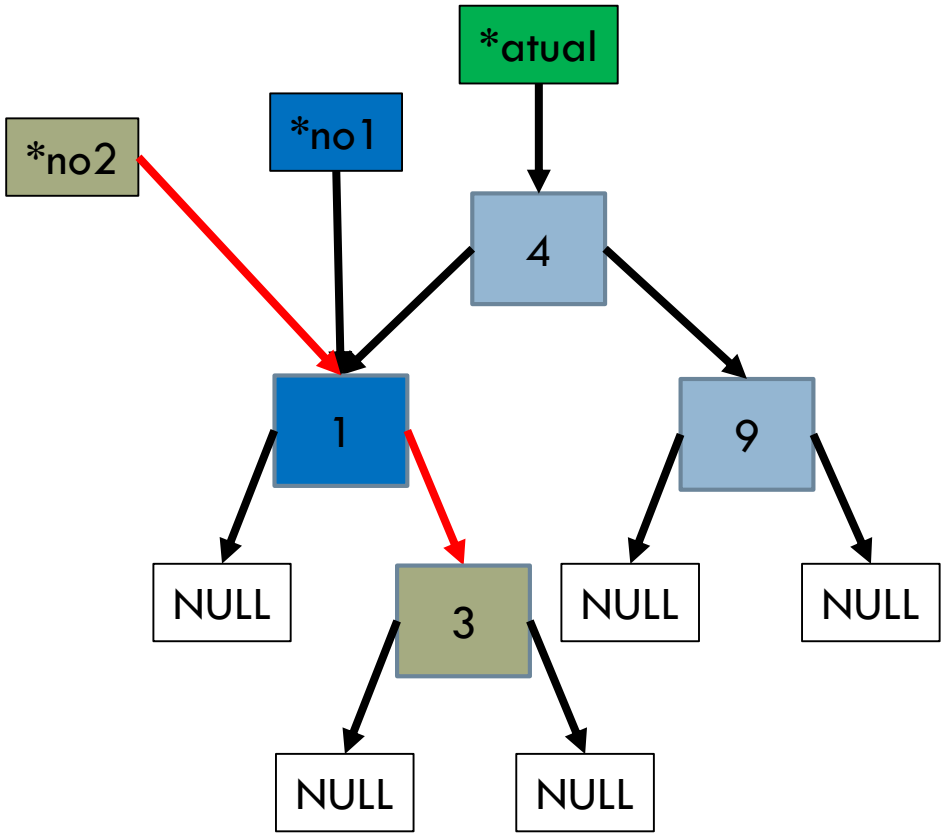
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



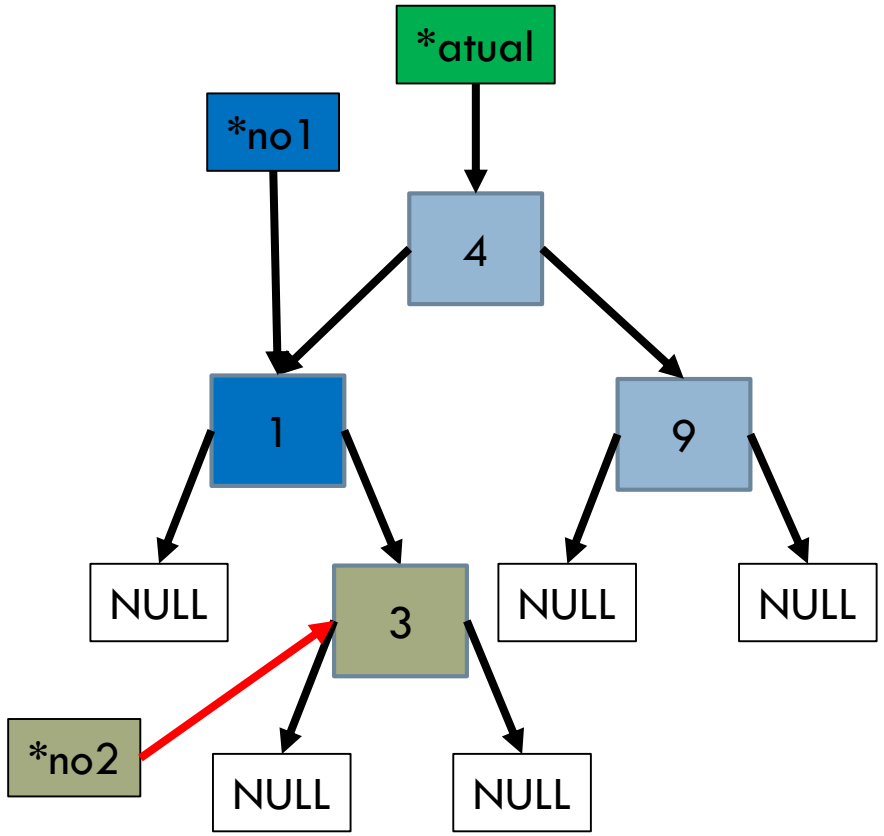
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



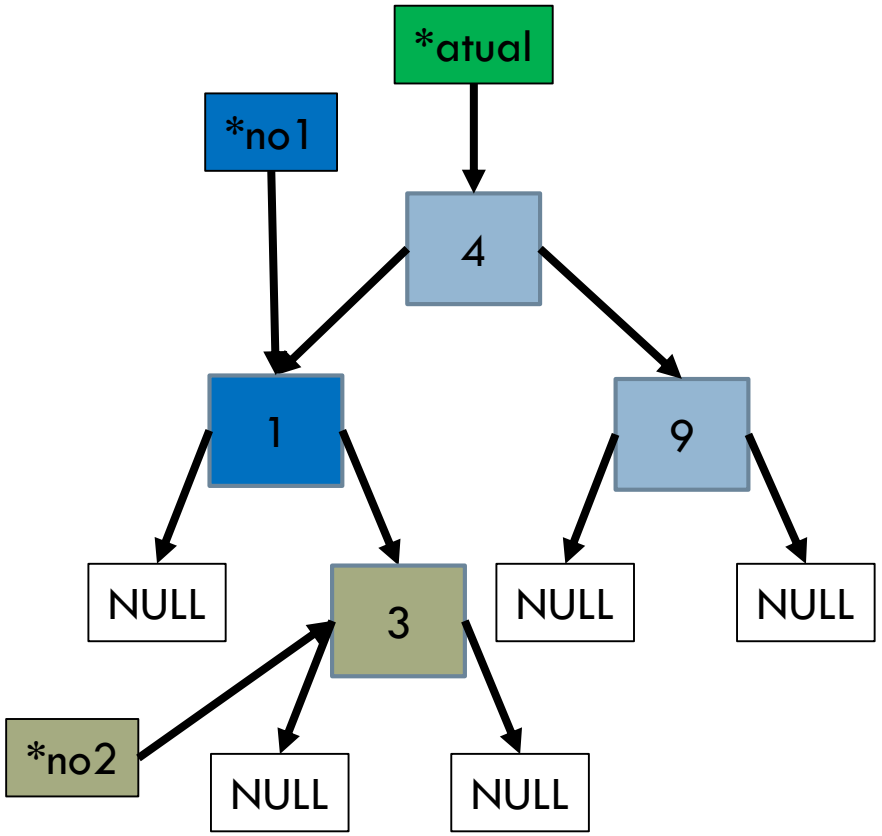
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



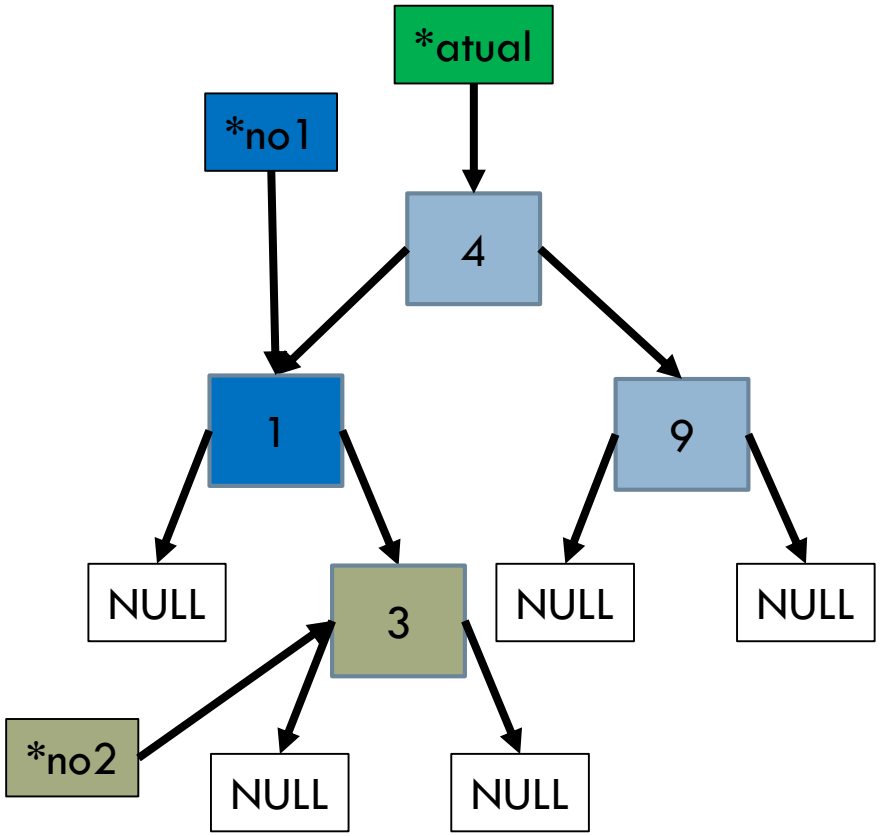
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



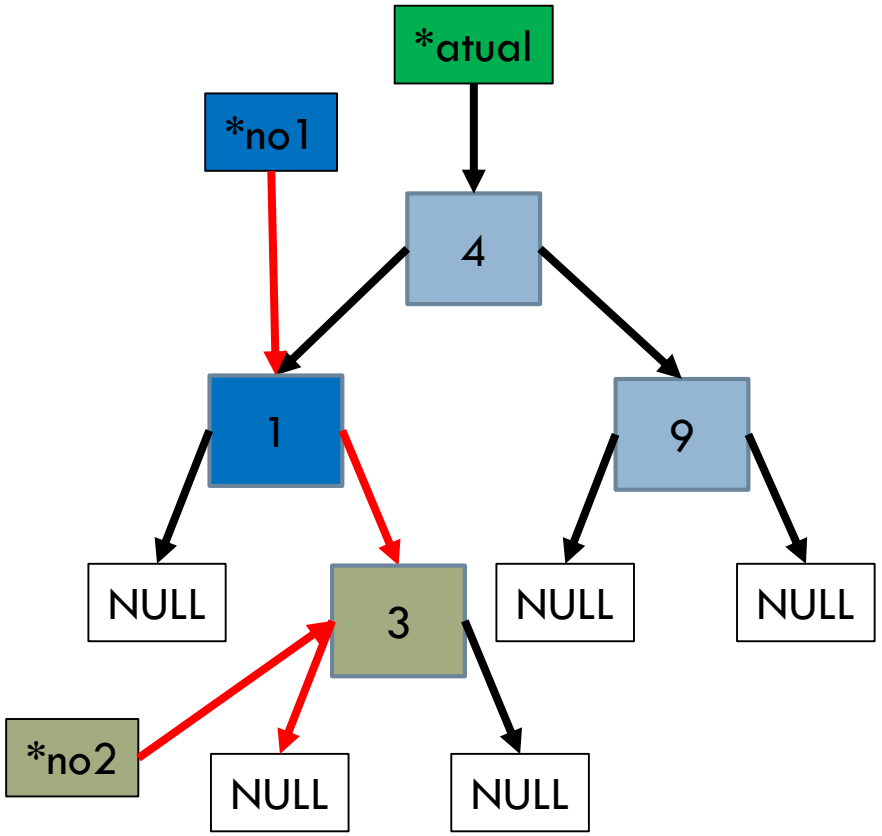
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



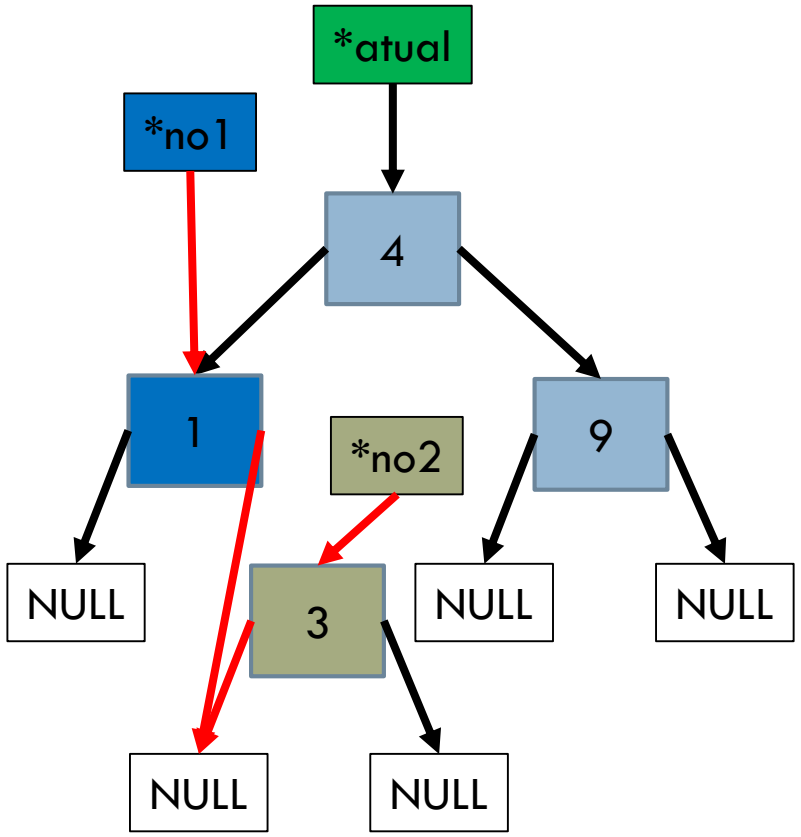
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



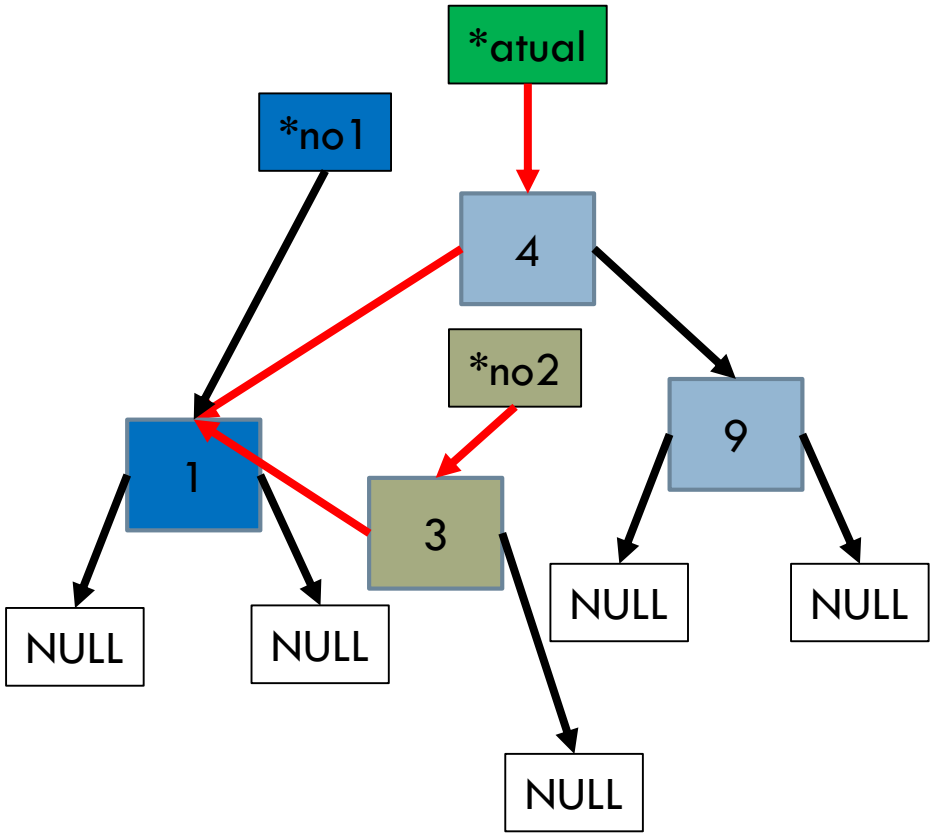
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



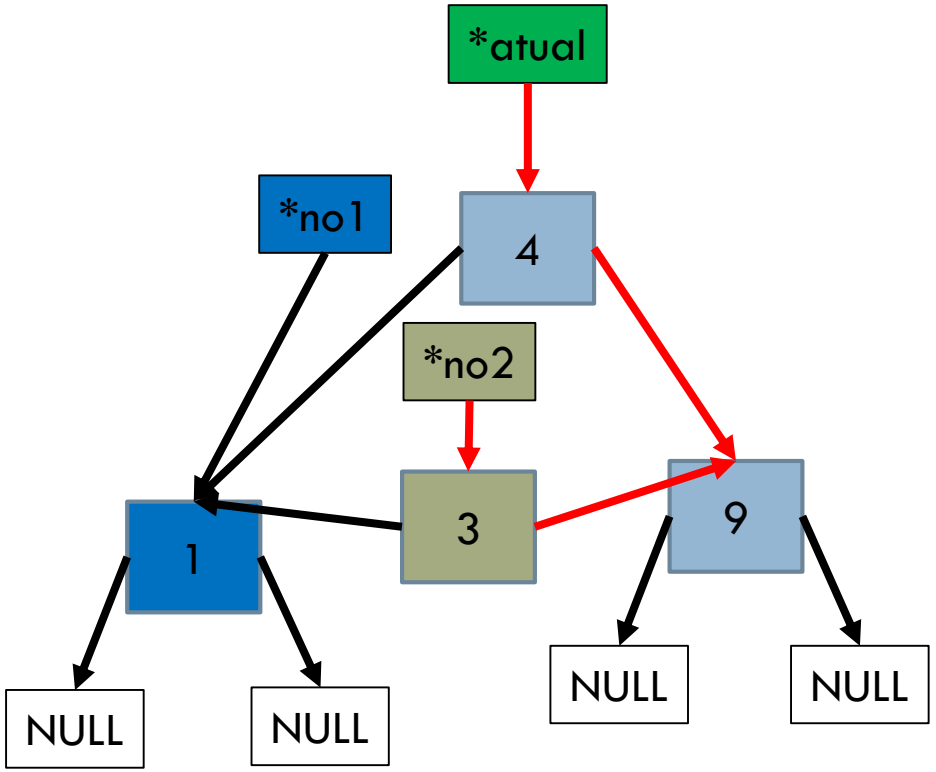
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



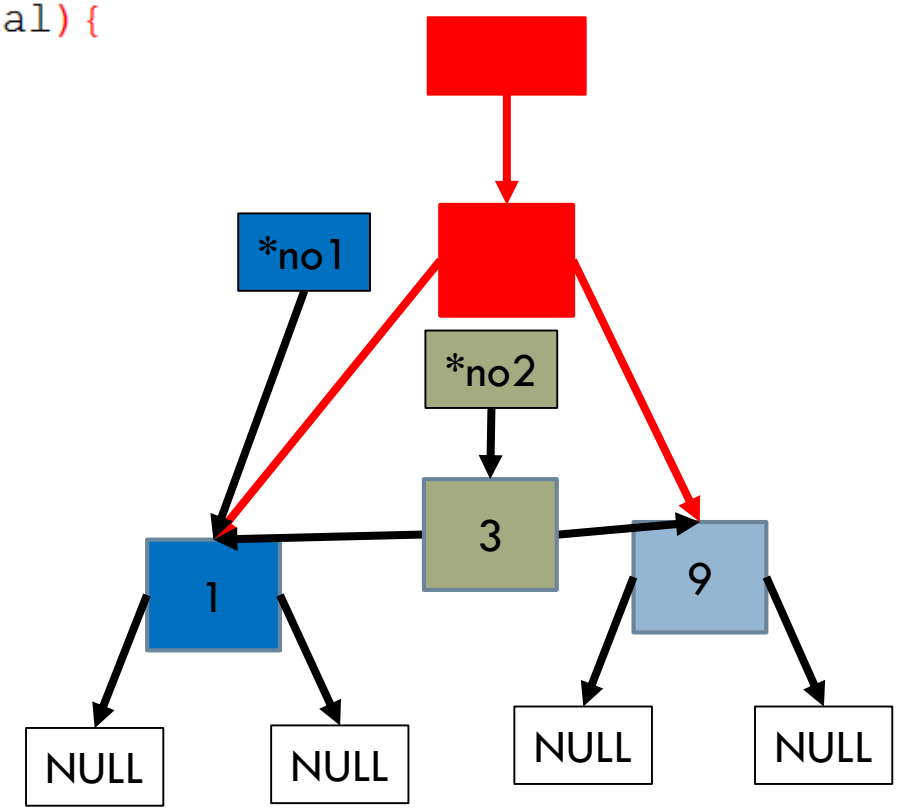
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



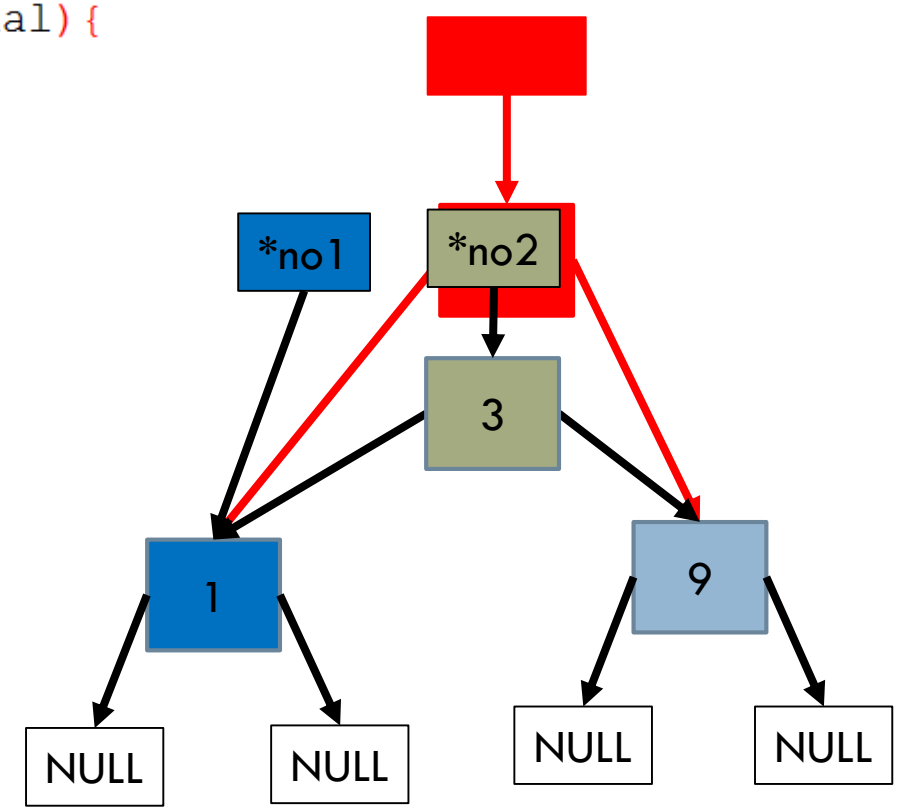
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



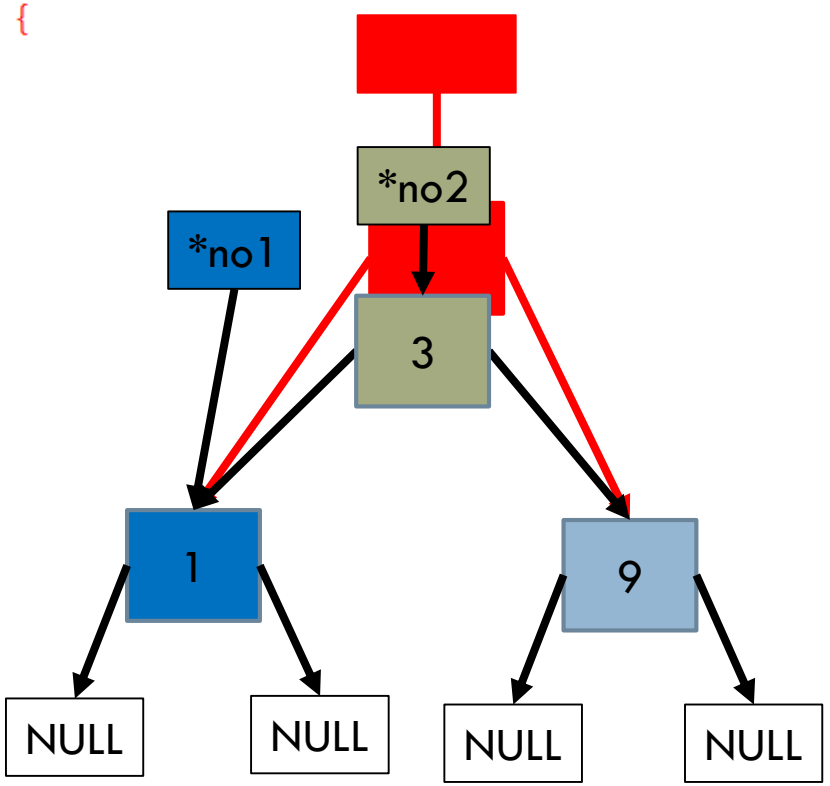
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



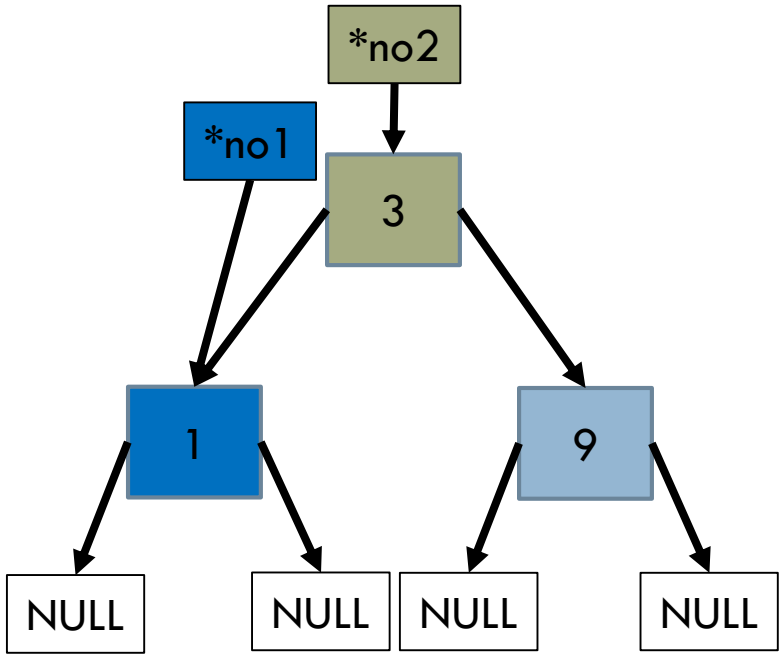
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



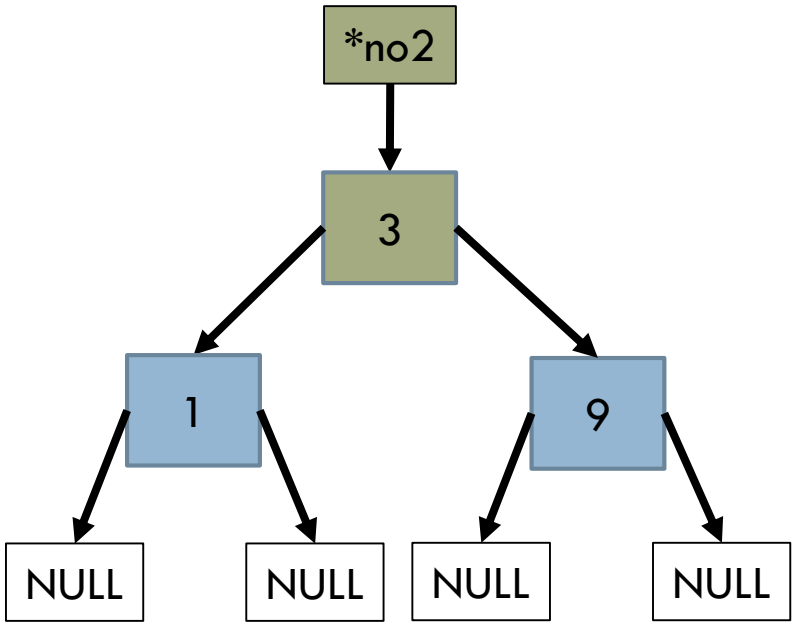
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



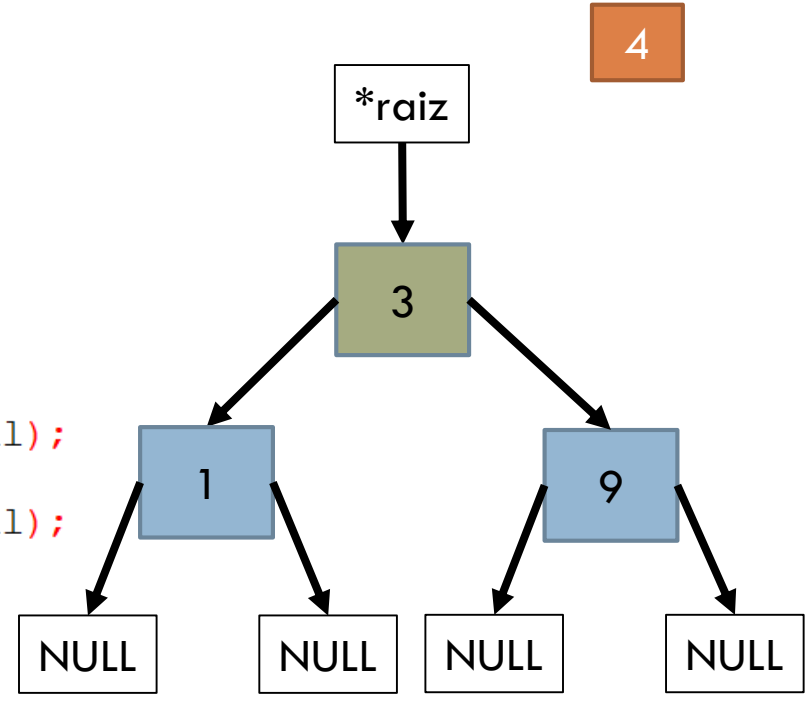
Remoção em ABB


```
struct NO* remove_atual(struct NO* atual){
    struct NO *no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        free(atual);
        return no2;
    }
    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }

    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```

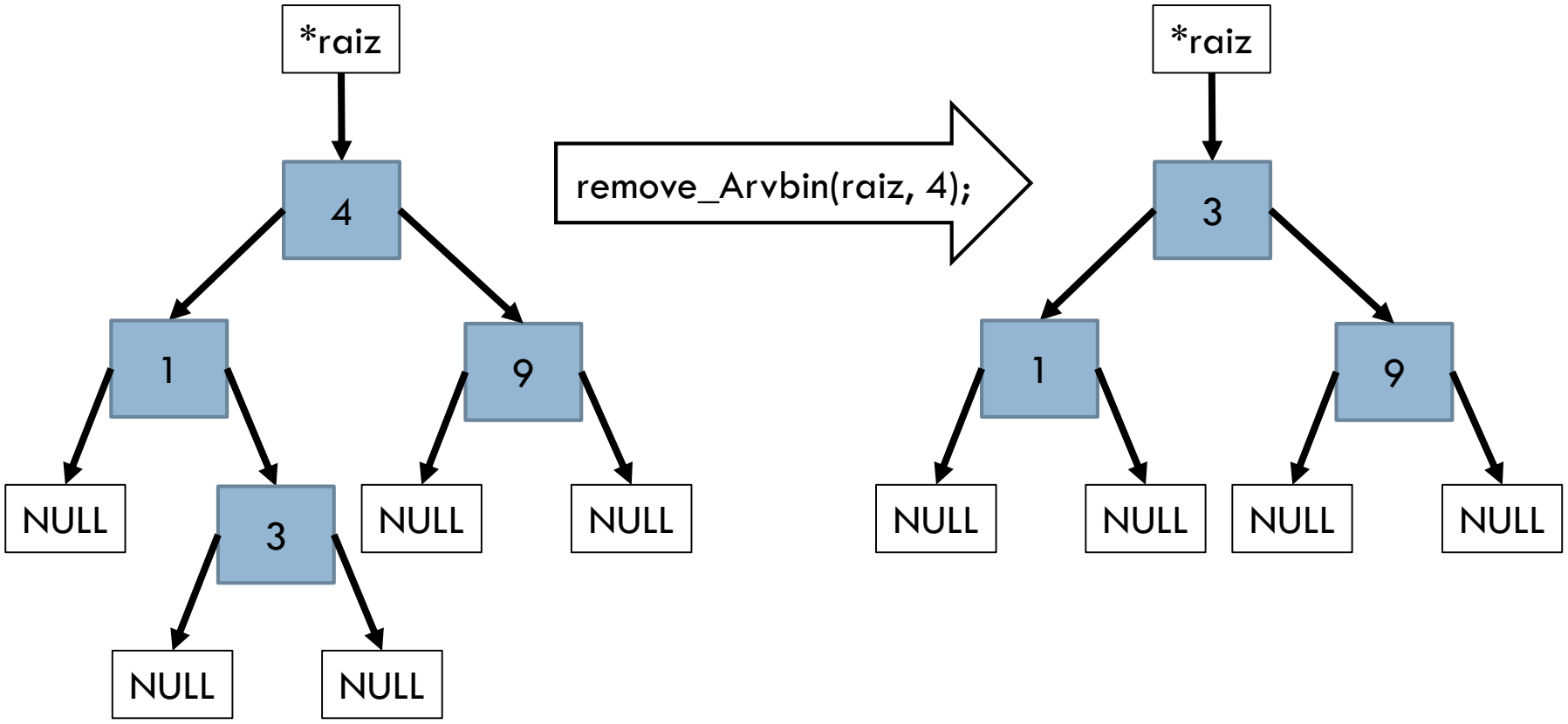


Remoção em ABB


```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = remove_atual(atual);
                else
                    ant->esq = remove_atual(atual);
            }
            return 1;
        }
        ant = atual;
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```



Remoção em ABB



Exercícios

- Baseado no código de exemplo de ABB disponível no moodle, faça:
  - Uma função para calcular a quantidade de nós de uma ABB.
  - Uma função para calcular a altura de uma ABB.
  - Uma função para imprimir o percurso Em-Ordem.
  - Uma função para imprimir o percurso Pós-Ordem.

Balanceamento de ABB

- Consiste na redistribuição dos nós para manter a árvore balanceada.

Balanceamento de ABB

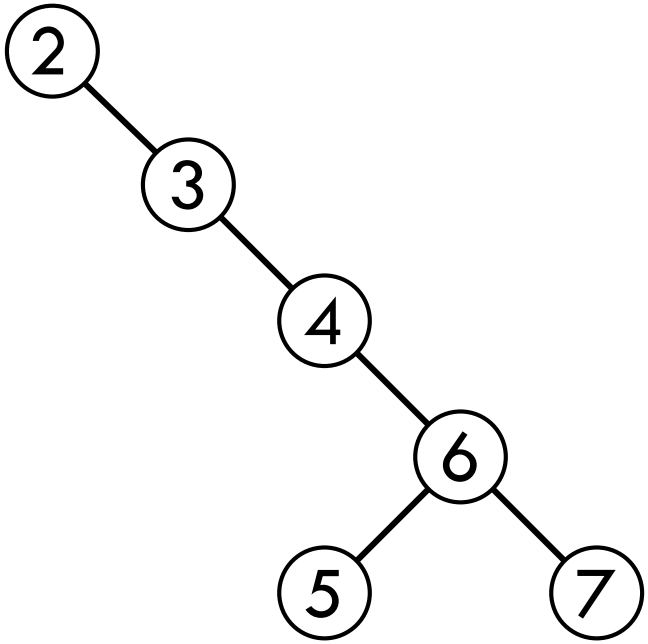
- Consiste na redistribuição dos nós para manter a árvore balanceada.
- Uma árvore balanceada é organizada para possuir a **menor altura** possível.
  - Se a diferença de altura entre as subárvores de qualquer nó é no máximo 1.
  - As *folhas* devem estar *no nível d ou d-1*, para uma árvore de *altura d*.

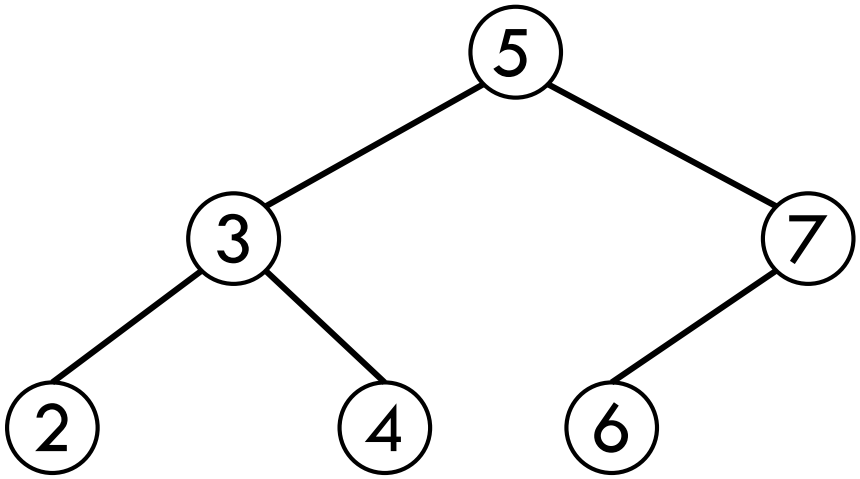
Balanceamento de ABB

- Consiste na redistribuição dos nós para manter a árvore balanceada.
- Uma árvore balanceada é organizada para possuir a **menor altura** possível.
  - Se a diferença de altura entre as subárvores de qualquer nó é no máximo 1.
  - As *folhas* devem estar *no nível d ou d-1*, para uma árvore de *altura d*.
- Quanto menor a altura da árvore, mais rápida será a busca por seus dados.

Balanceamento de ABB

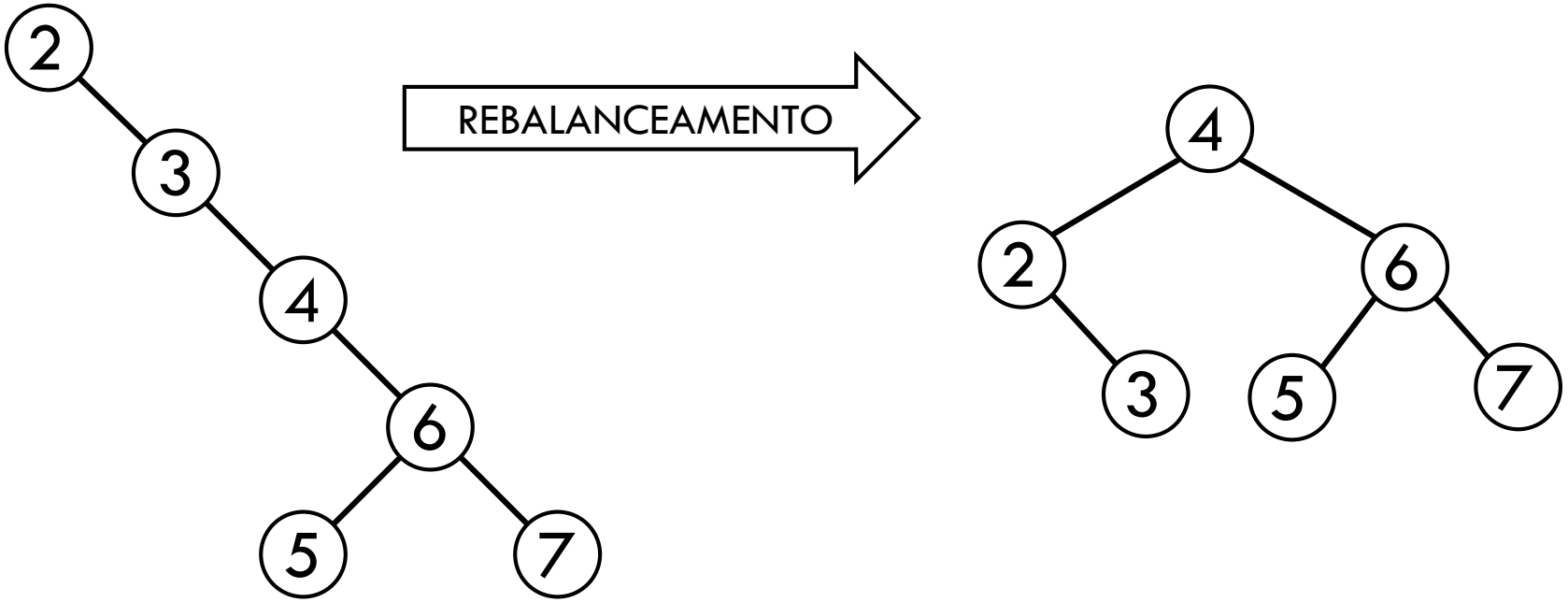
- Exemplos de árvores: 2-3-4-5-6-7





Rebalanceamento de ABB

- Reorganizar uma ABB para ficar balanceada.

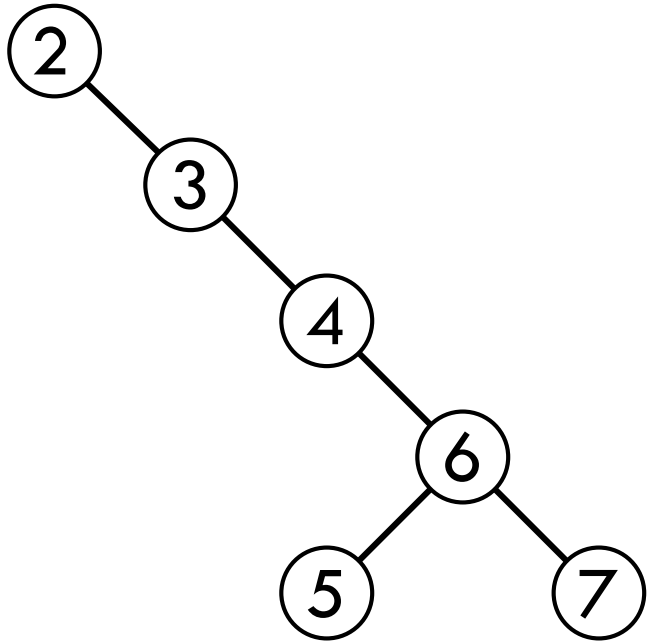


Rebalanceamento de ABB

- Algoritmo:
  - Salvar dados em um vetor auxiliar, utilizando o percurso Em-Ordem;
  - Inserir na ABB o registro do meio do vetor.
    - Repetir para intervalo do lado esquerdo;
    - Repetir para intervalo do lado direito.
  - Repetir até inserir todos os elementos.

Rebalanceamento de ABB

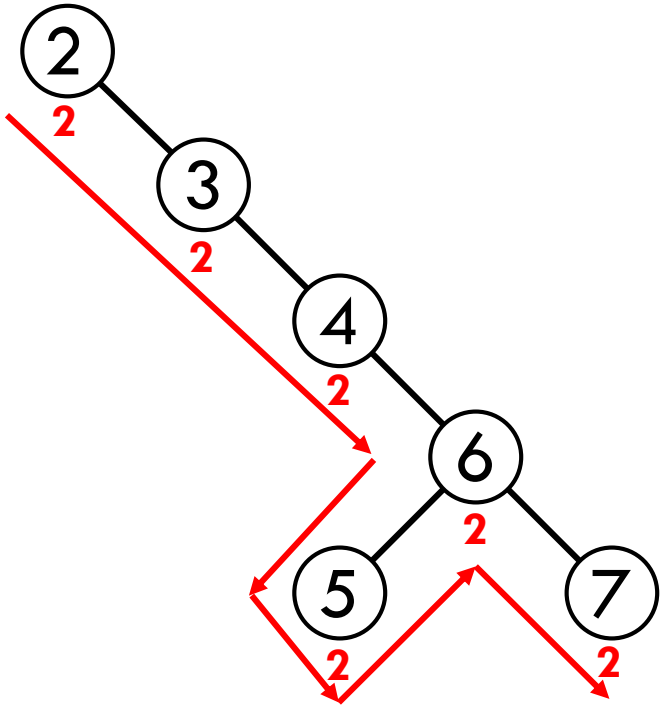
- Salvar dados em um vetor auxiliar, utilizando o percurso Em-Ordem:



0	1	2	3	4	5

Rebalanceamento de ABB

- Salvar dados em um vetor auxiliar, utilizando o percurso Em-Ordem:



0	1	2	3	4	5
2	3	4	5	6	7

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	
FIM	
MEIO	

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	**2**

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

④

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	**2**

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

④

	ÍNDICE
INÍCIO	0
FIM	1
MEIO	

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

④

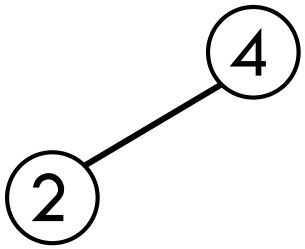
	ÍNDICE
INÍCIO	0
FIM	1
MEIO	**0**

Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	0
FIM	1
MEIO	**0**

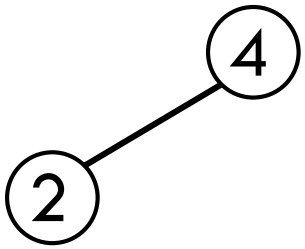


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	

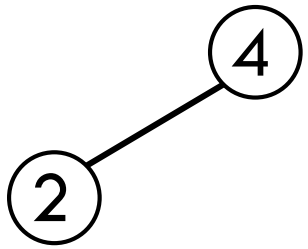


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	**1**

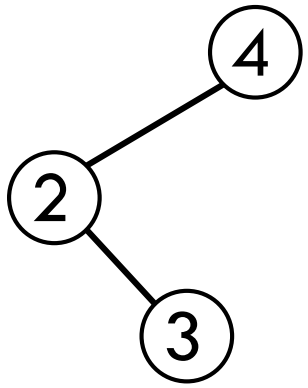


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	**1**

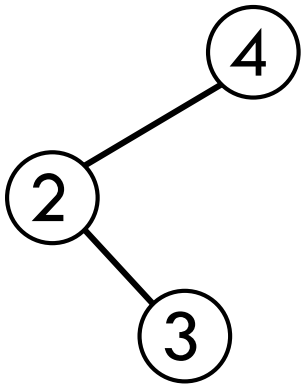


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	

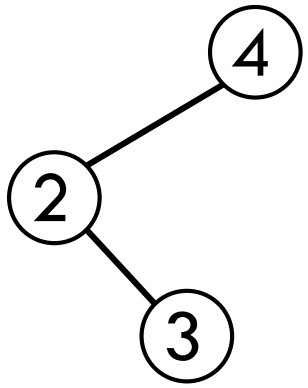


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	**4**

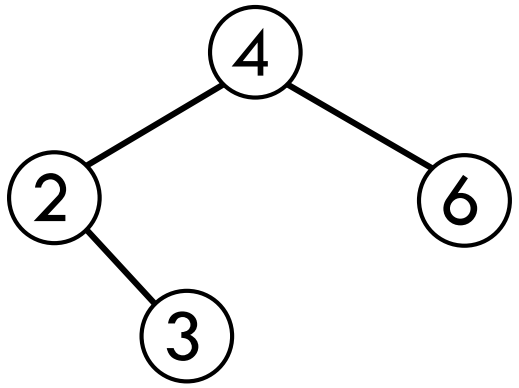


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	**4**

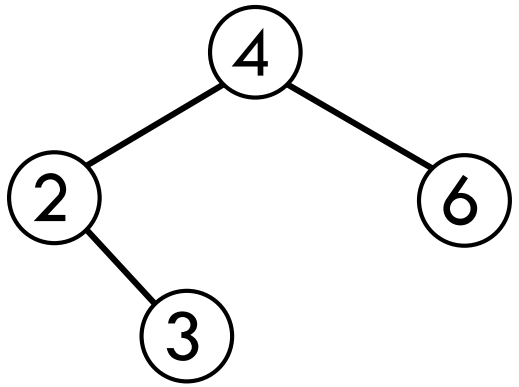


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	

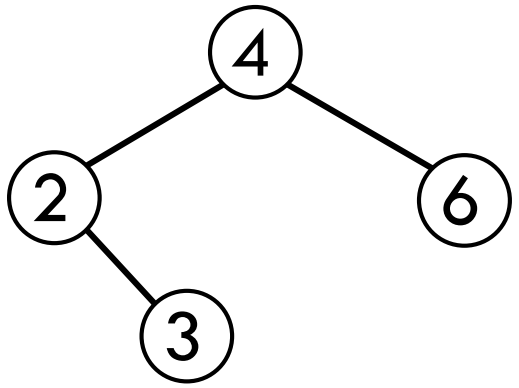


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	**3**

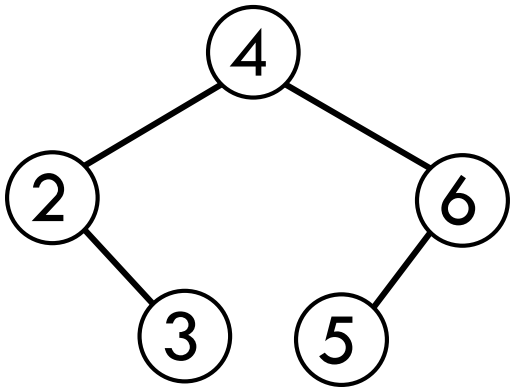


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	**3**

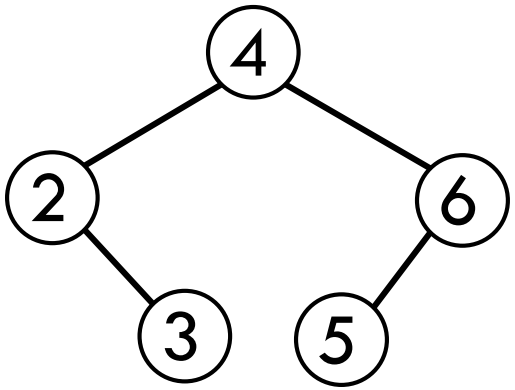


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	*5*
FIM	*5*
MEIO	

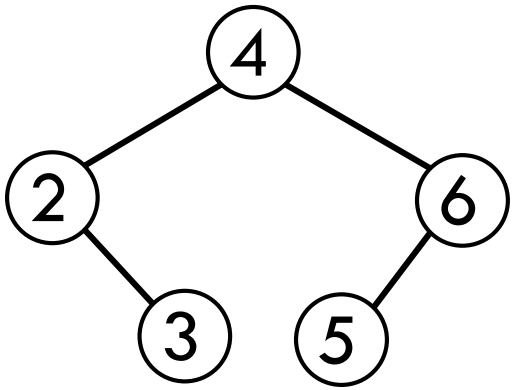


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	*5*
FIM	*5*
MEIO	**5**

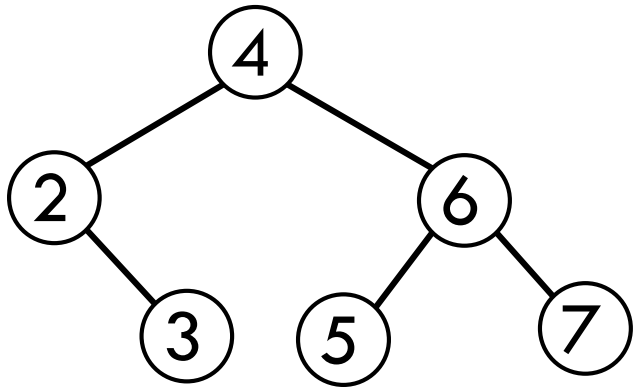


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	5
FIM	5
MEIO	**5**

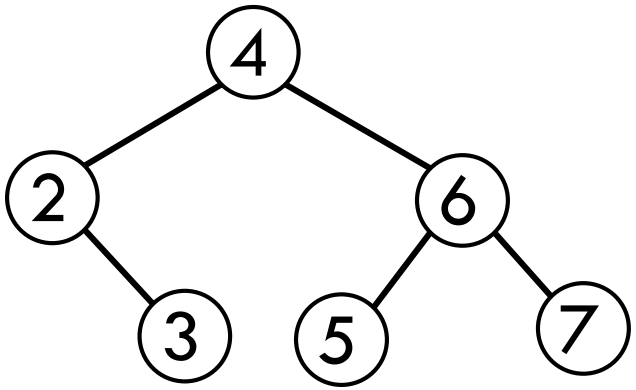


Rebalanceamento de ABB

- Inserir o registro do meio do vetor.
  - Repetir para intervalo do lado esquerdo;
  - Repetir para intervalo do lado direito.
- Meio = (Início + Fim)/2

0	1	2	3	4	5
2	**3**	**4**	**5**	**6**	**7**

	ÍNDICE
INÍCIO	-
FIM	-
MEIO	**-**



Exercício

- Faça a inserção dos dados:
  - 1-2-3-4-5-6-7-8-9-10-11
- Realize o rebalanceamento da árvore conforme o algoritmo anterior.