

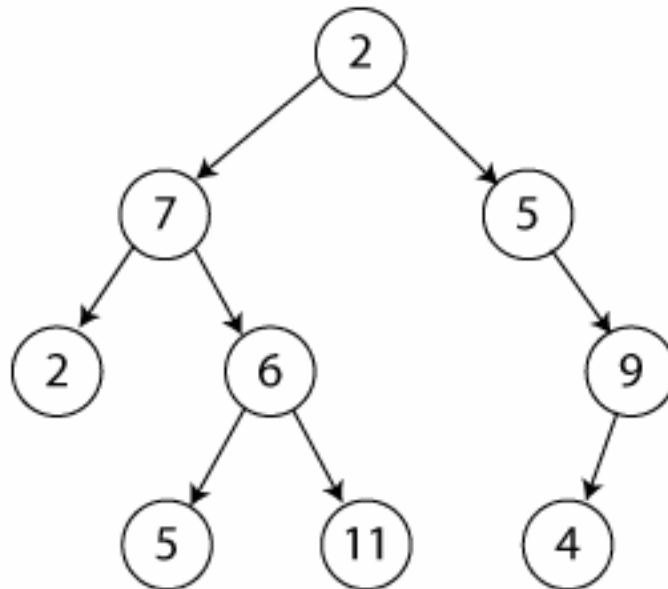
ÁRVORES BINÁRIAS DE BUSCA

Prof. Muriel Mazzetto
Estrutura de Dados

Árvore binária

2

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



Árvore binária de busca

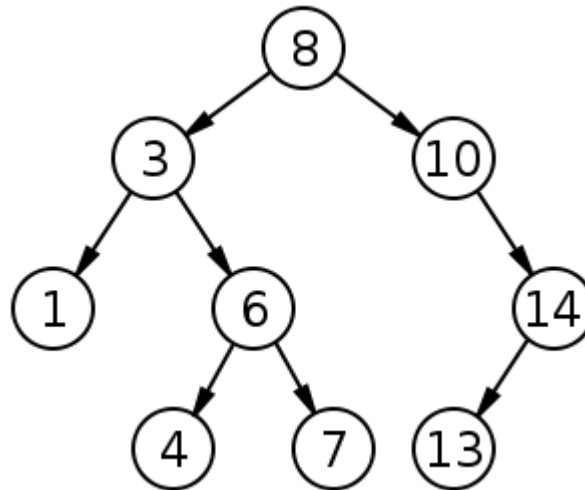
3

- 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

4

- 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

5

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

Tipos de dados da ABB

6

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

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

//Arquivo .h
typedef struct NO* ArvBin;

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

Criação da ABB

7

- ❑ 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

8

- 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

9

- 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

10

- 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

11

- 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

12

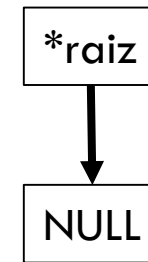
- Algoritmo `insere(*raiz, dado)`:
 - ▣ Se posição está vazia (NULL), insere o dado;
 - ▣ Se é menor que raiz, `insere((*raiz)->esquerda, dado)`;
 - ▣ Se é maior que raiz, `insere((*raiz)->direita, dado)`;
 - ▣ Se é igual, descarta o dado.

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

Inserção em ABB

13

```
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;  
    [...]
```



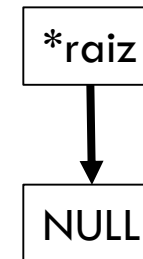
4

Inserção em ABB

14

```
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



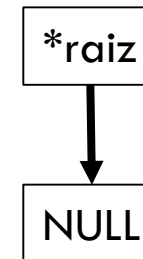
Inserção em ABB

15

```
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



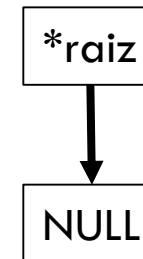
novo

Inserção em ABB

16

```
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;  
    [...]
```

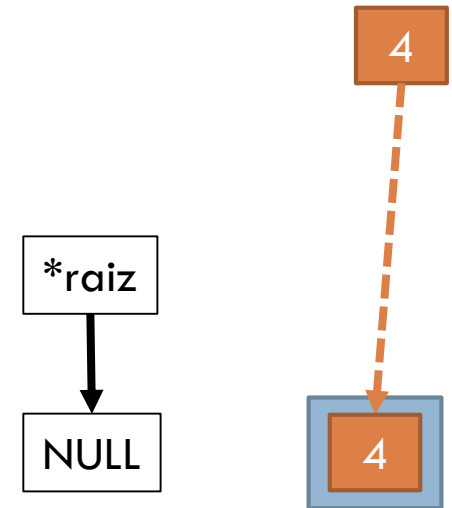
4



Inserção em ABB

17

```
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;  
    [...]
```

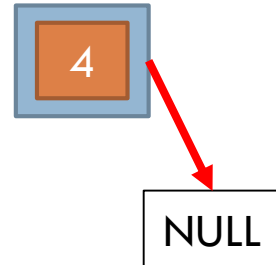
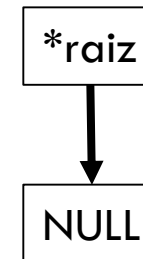


Inserção em ABB

18

```
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

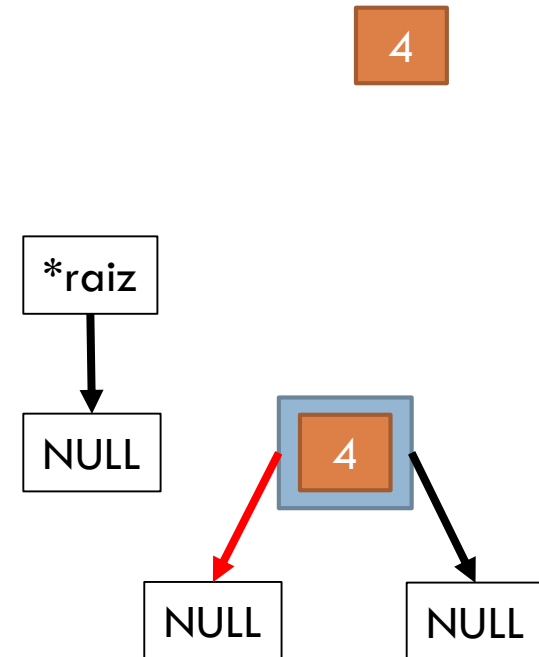


Inserção em ABB

19

```
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;

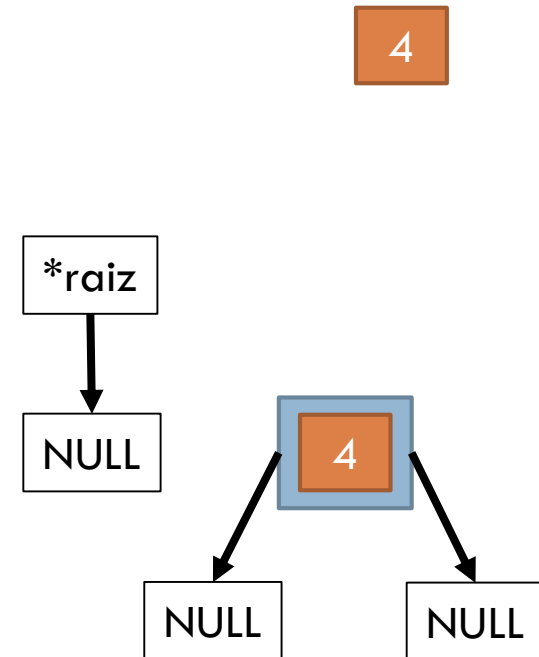
    if(*raiz == NULL)
        *raiz = novo;
    [...]
```



Inserção em ABB

20

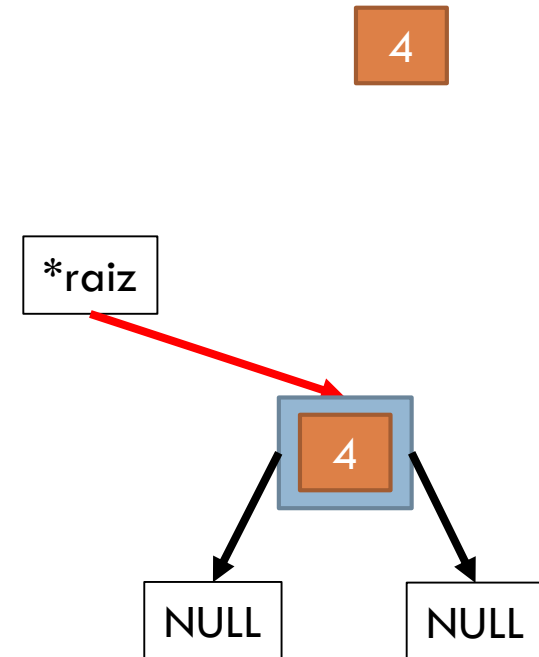
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
    → if(*raiz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



Inserção em ABB

21

```
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;  
    [...]
```



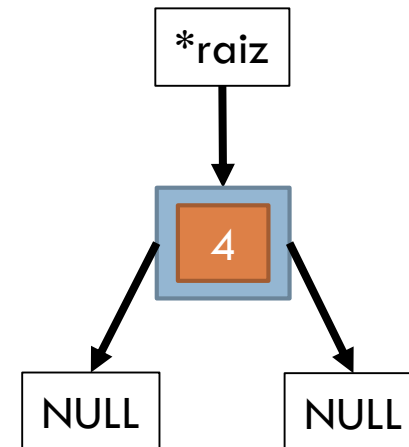
Inserção em ABB

22

```
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
        }

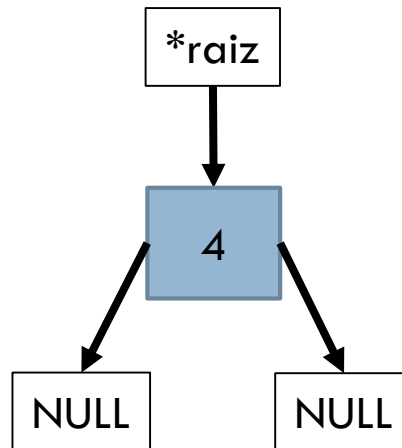
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
→ return 1;
}
```

4



Inserção em ABB

23

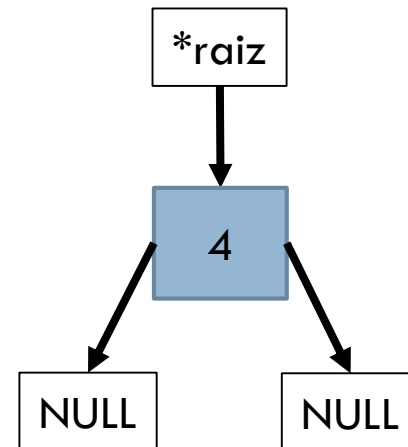


Inserção em ABB

24

```
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

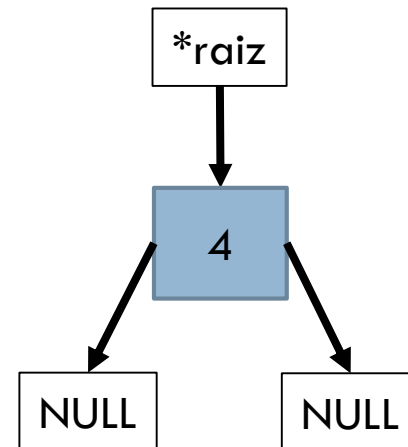


Inserção em ABB

25

```
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

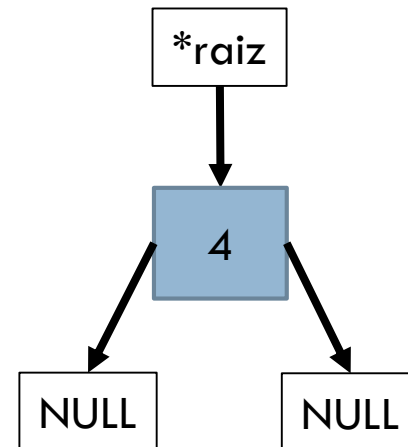


Inserção em ABB

26

```
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

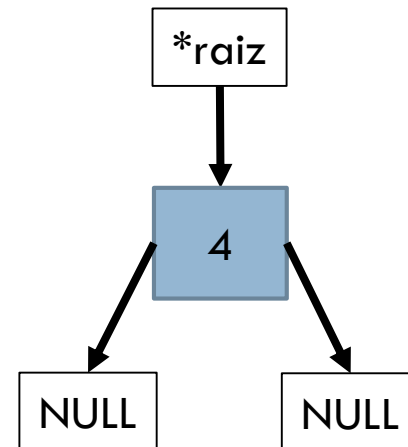


Inserção em ABB

27

```
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;  
    [...]
```

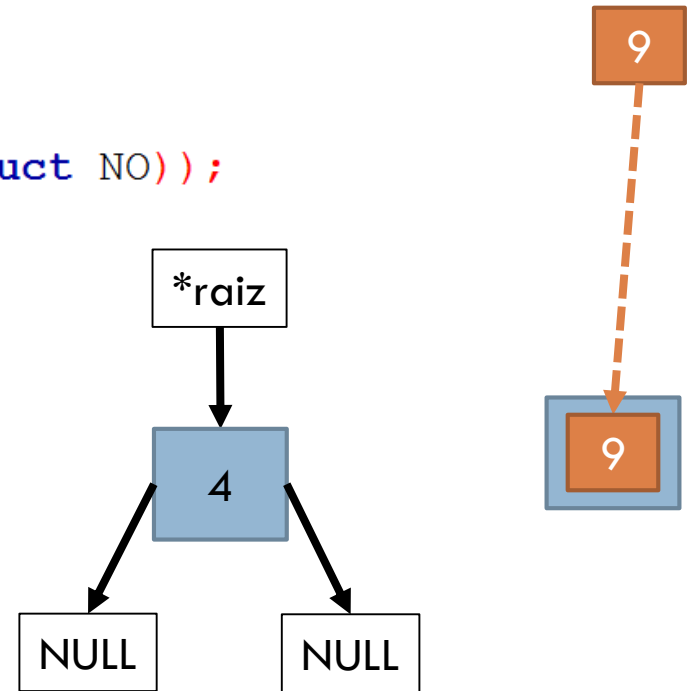
9



Inserção em ABB

28

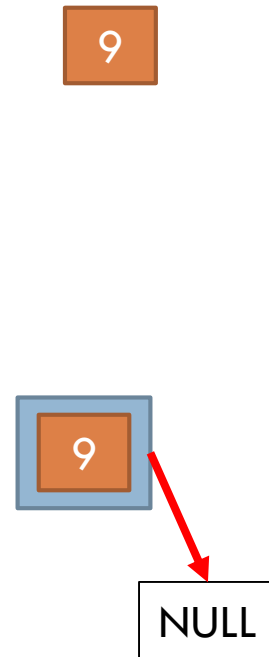
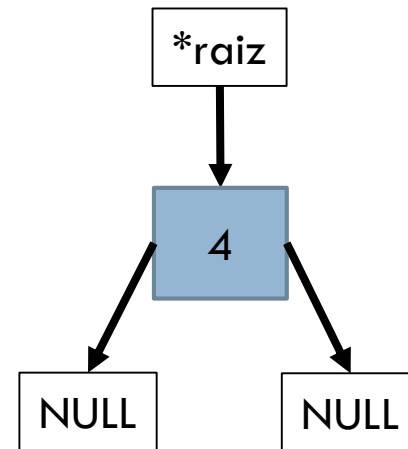
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    → novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

29

```
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;  
    [...]
```

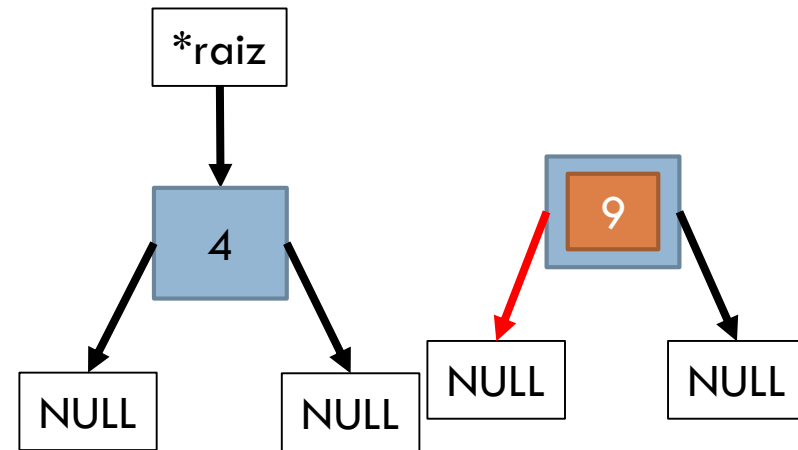


Inserção em ABB

30

```
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

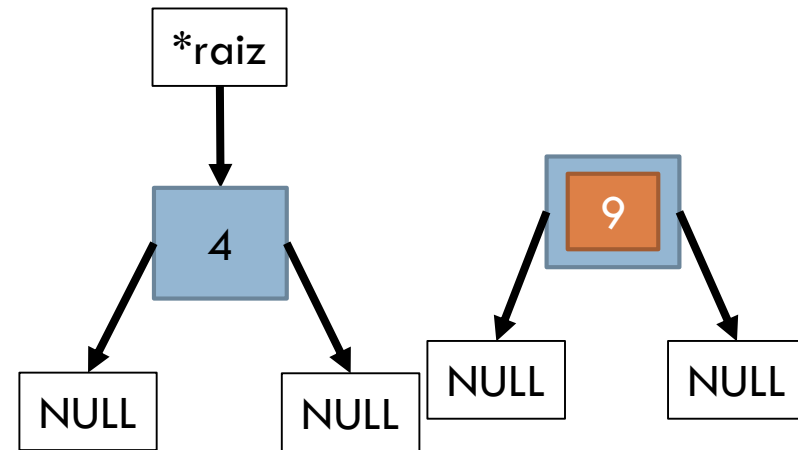


Inserção em ABB

31

```
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



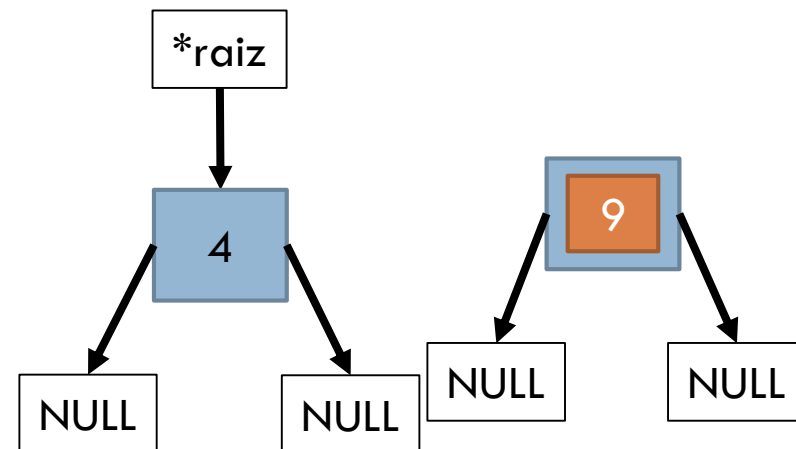
Inserção em ABB

32

```
→ 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

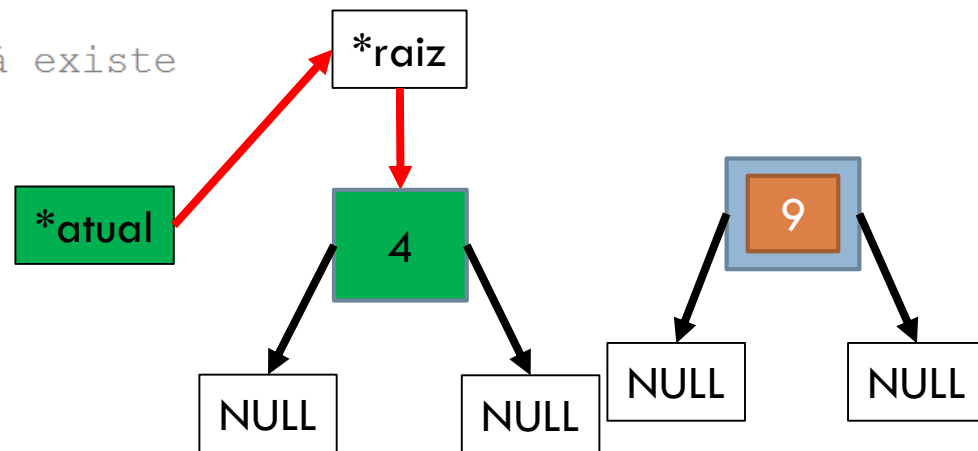


Inserção em ABB

33

```
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

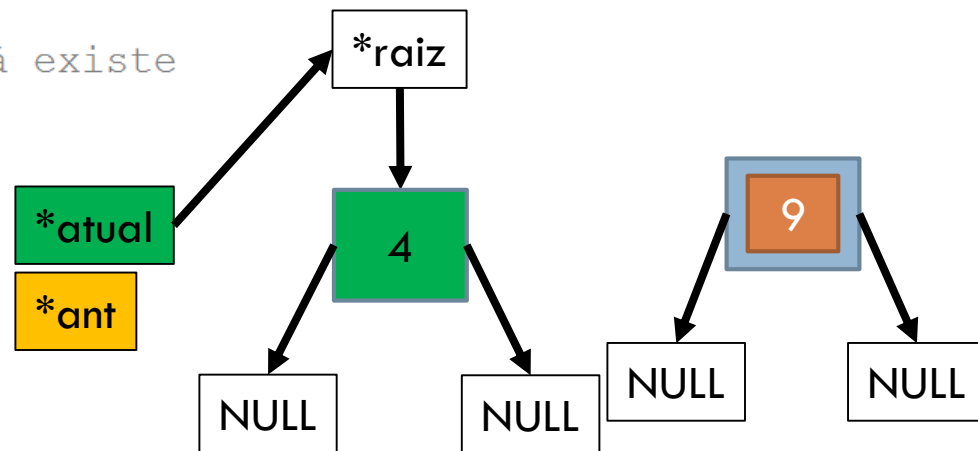


Inserção em ABB

34

```
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



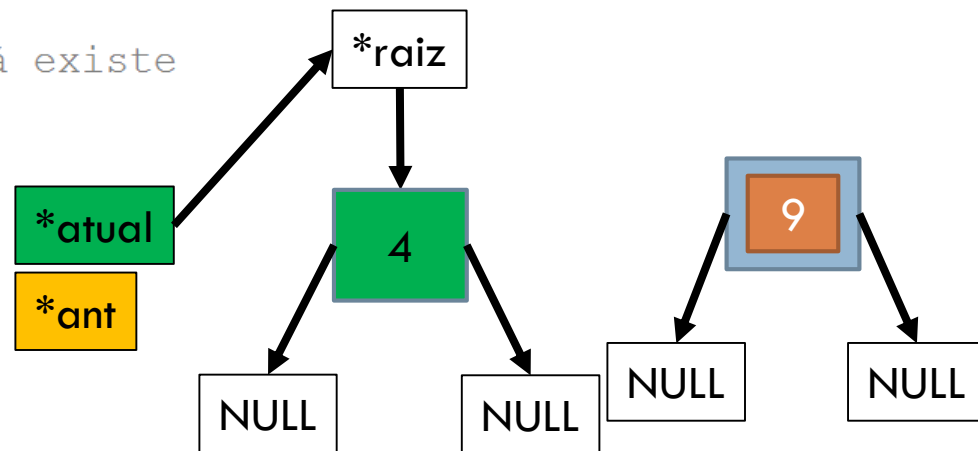
Inserção em ABB

35

```
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



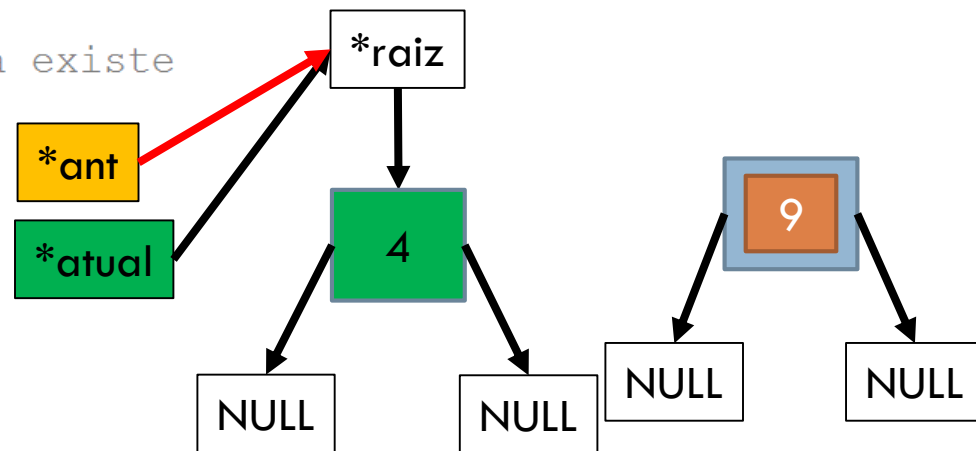
Inserção em ABB

36

```
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



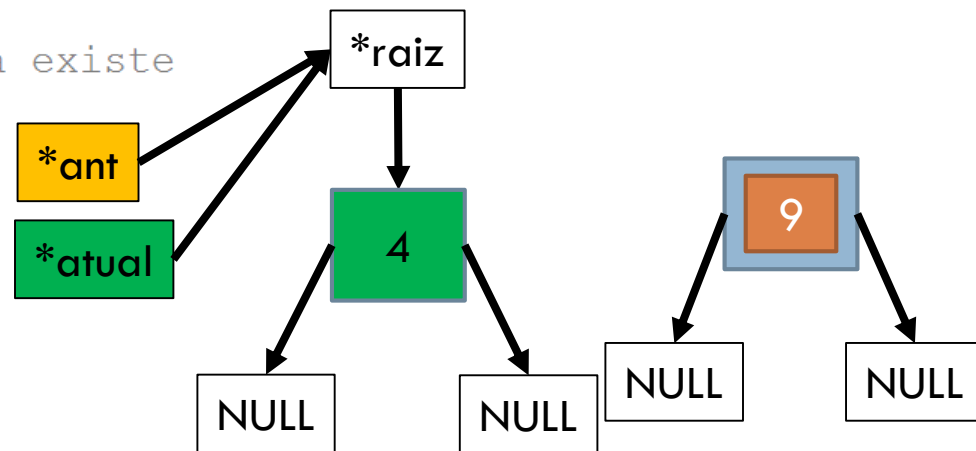
Inserção em ABB

37

```
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;
}
```

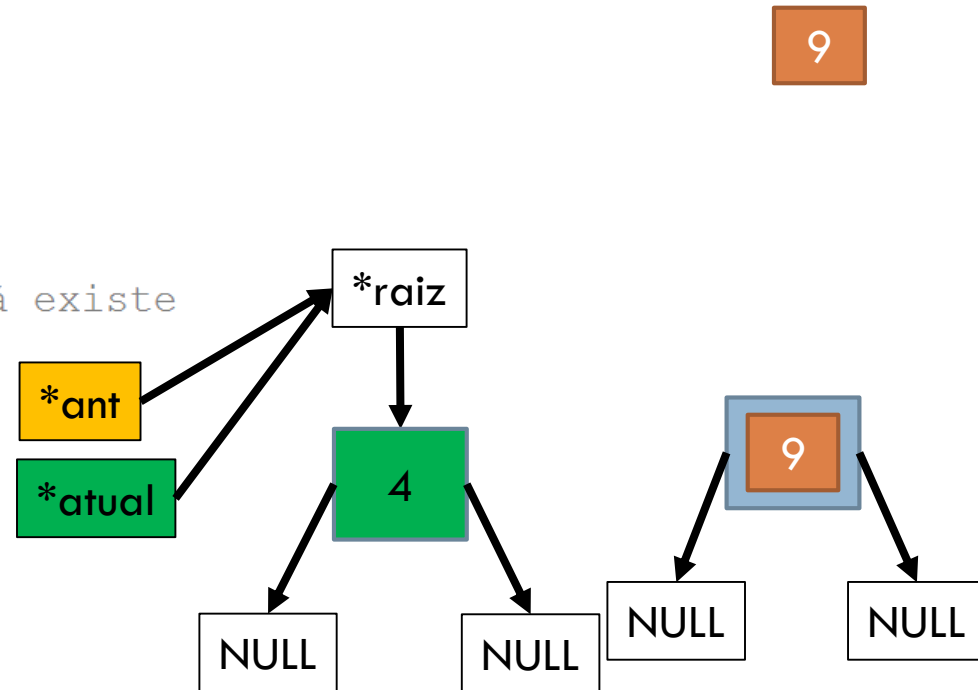
9



Inserção em ABB

38

```
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;  
}
```

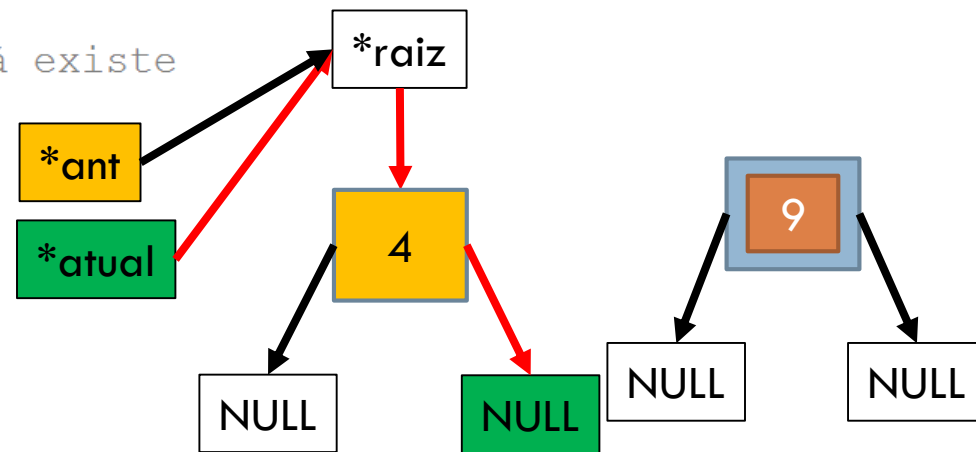


Inserção em ABB

39

```
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;  
}
```

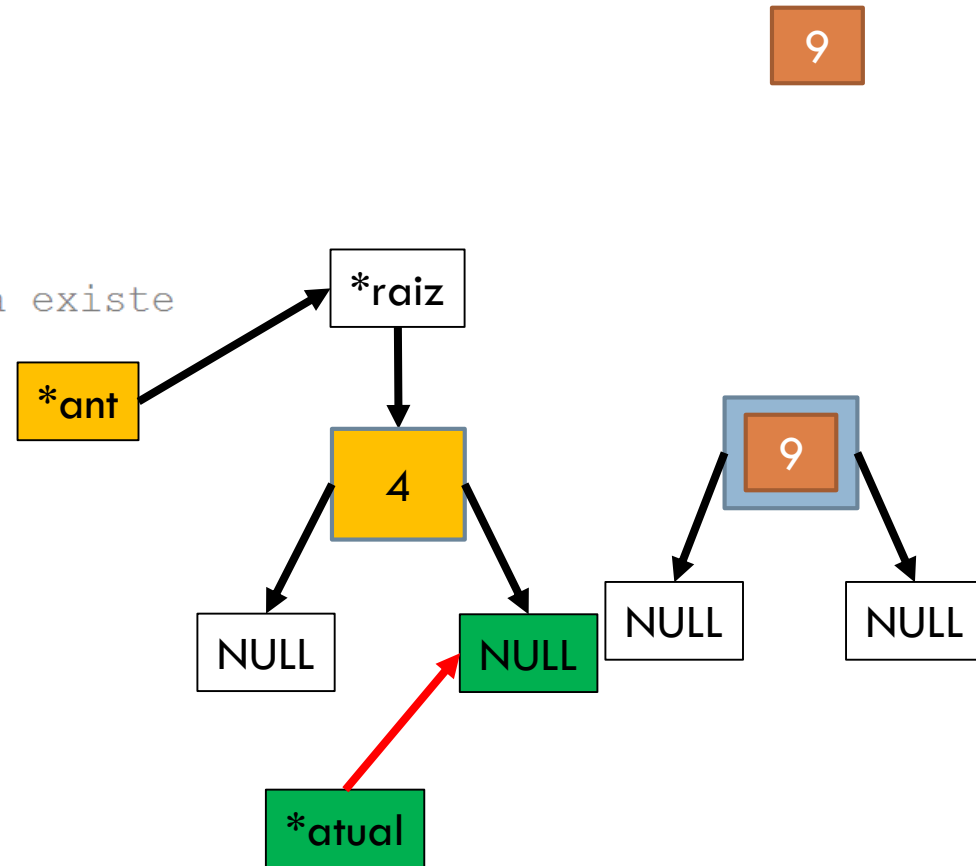
9



Inserção em ABB

40

```
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;  
}
```

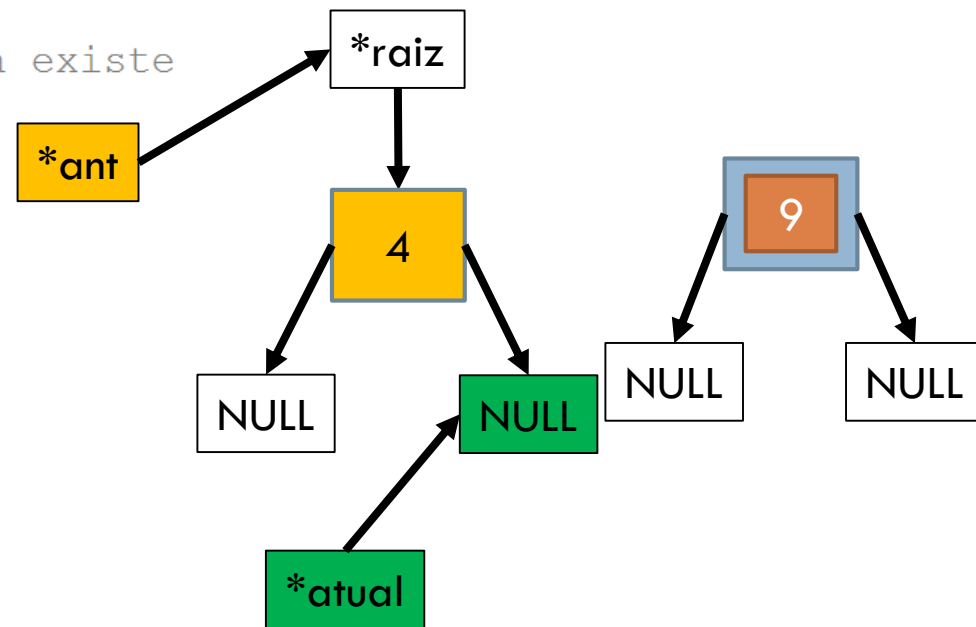


Inserção em ABB

41

```
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;
```

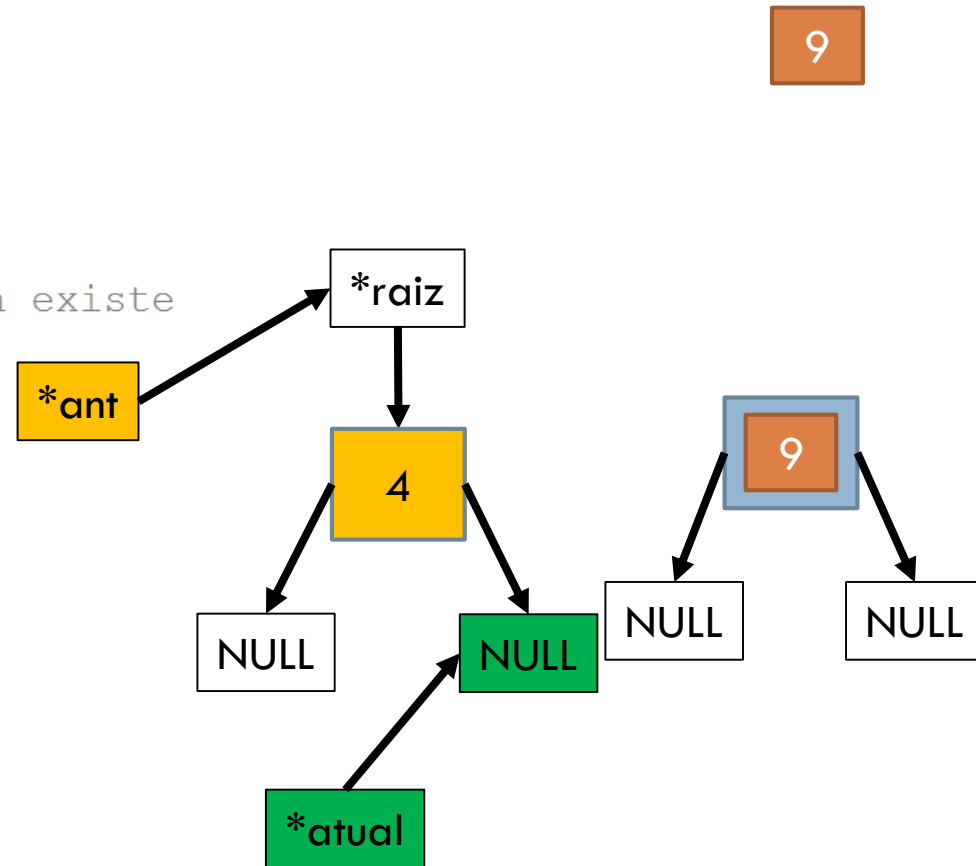
9



Inserção em ABB

42

```
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;  
}
```

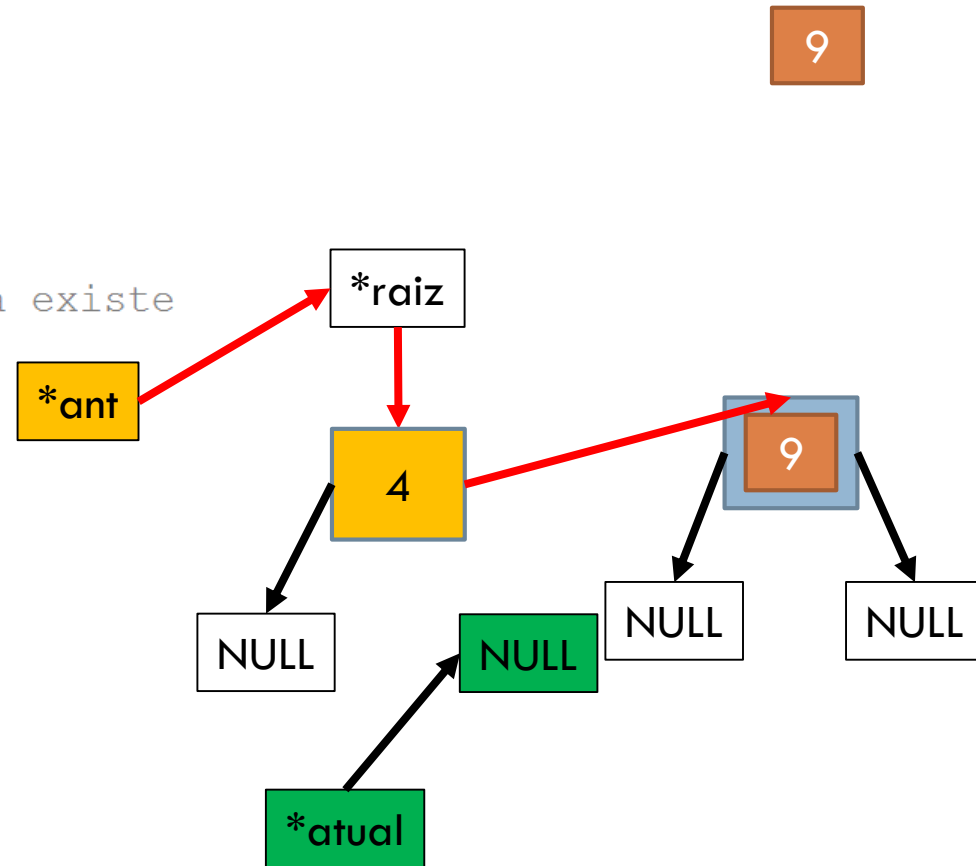


Inserção em ABB

43

```
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;
}
```

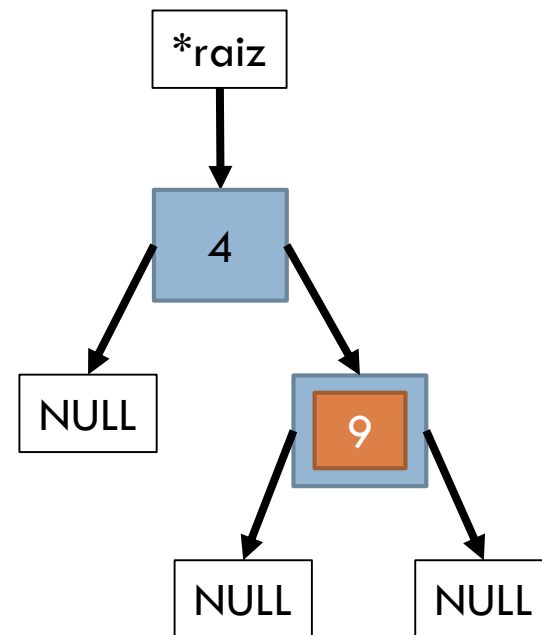


Inserção em ABB

44

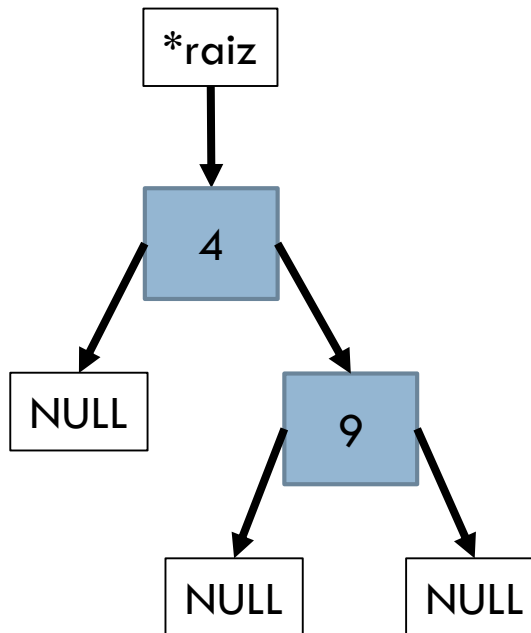
```
else{  
    [...]  
    struct NO* atual = *raiz;  
    struct NO* ant = NULL;  
    while(atual != NULL){  
        ant = atual;  
        if(valor == atual->info){  
            free(novo);  
            return 0; //elemento já existe  
        }  
  
        if(valor > atual->info)  
            atual = atual->dir;  
        else  
            atual = atual->esq;  
    }  
    if(valor > ant->info)  
        ant->dir = novo;  
    else  
        ant->esq = novo;  
    }  
    return 1;  
}
```

9



Inserção em ABB

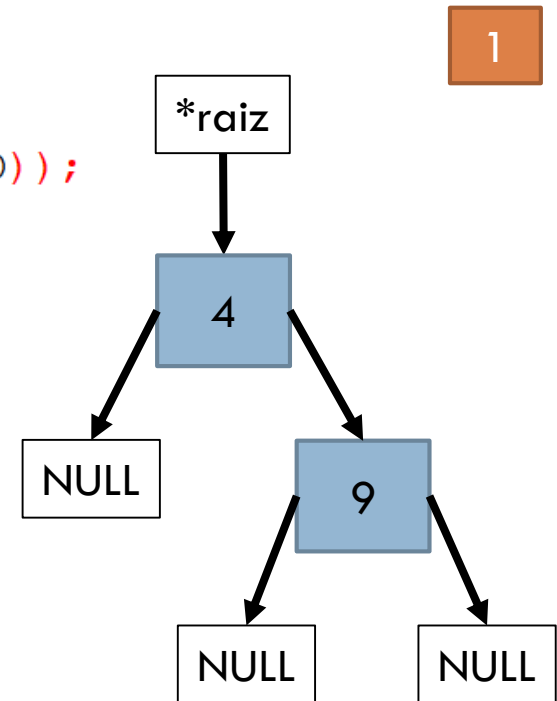
45



Inserção em ABB

46

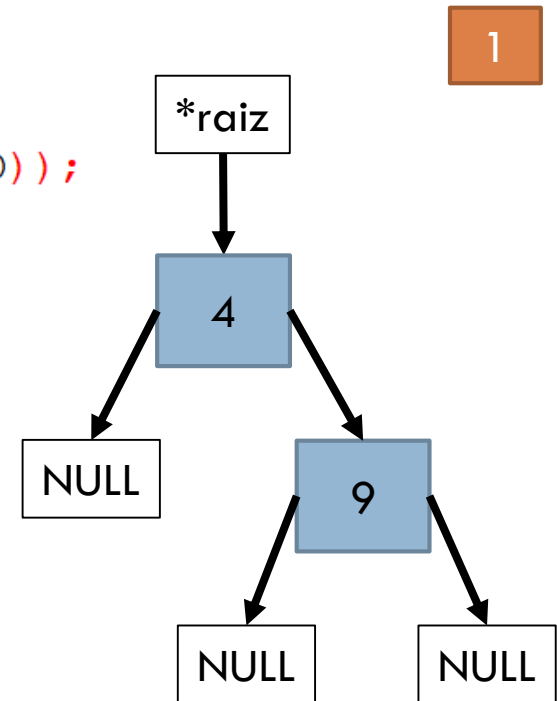
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

47

```
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;  
    [...]
```

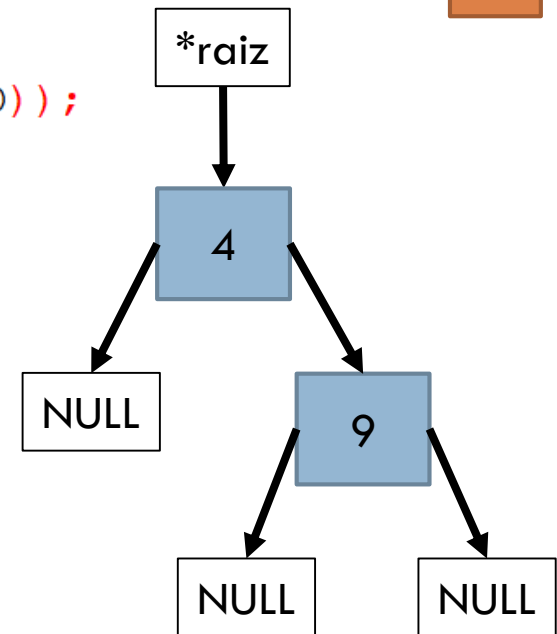


Inserção em ABB

48

```
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;  
      [...]
```

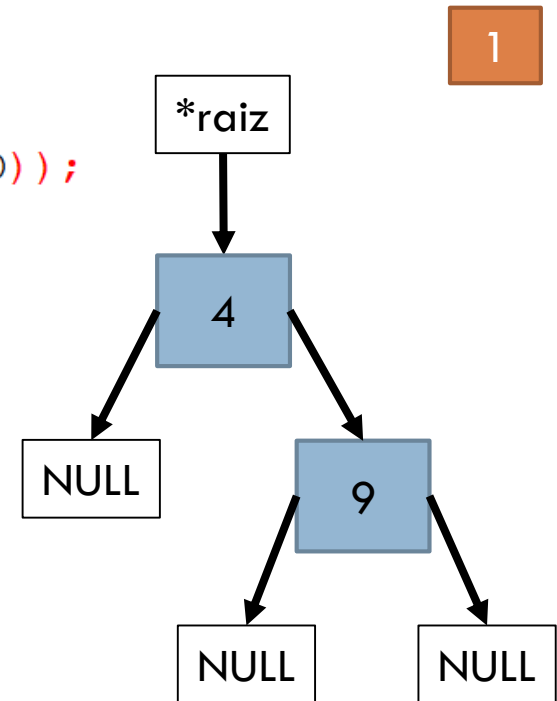
novo



Inserção em ABB

49

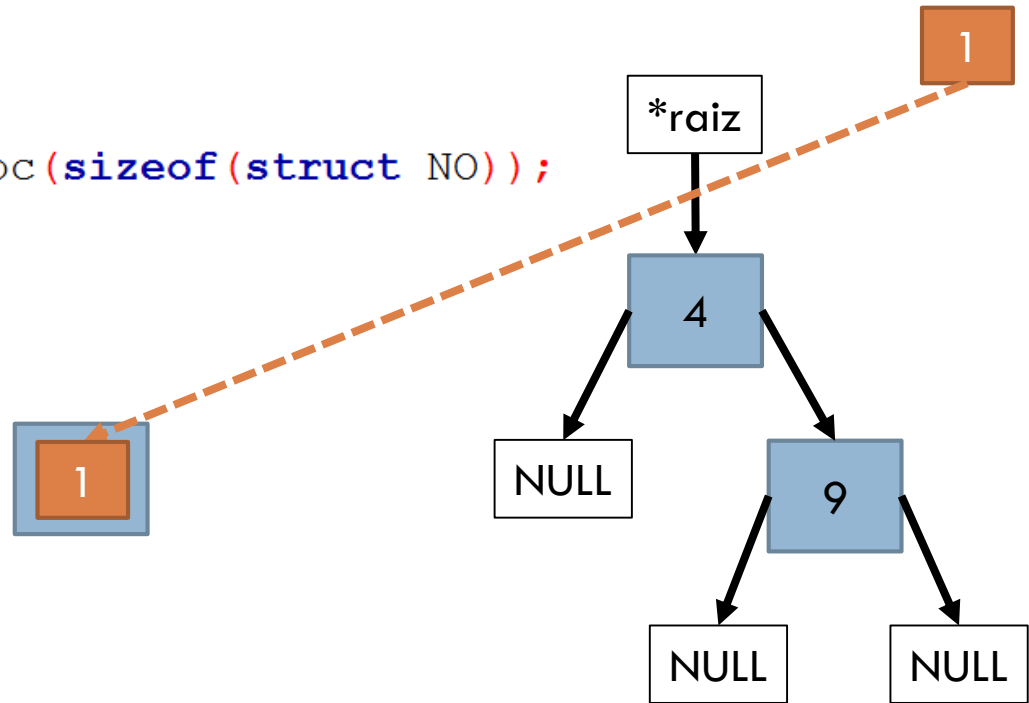
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    → if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

50

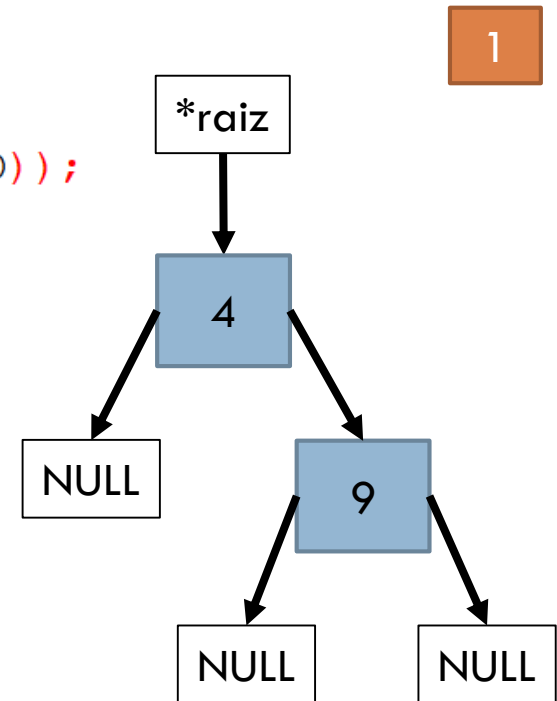
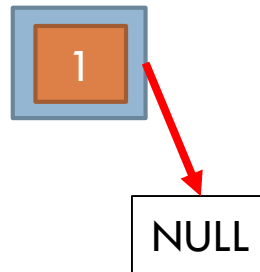
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    → novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

51

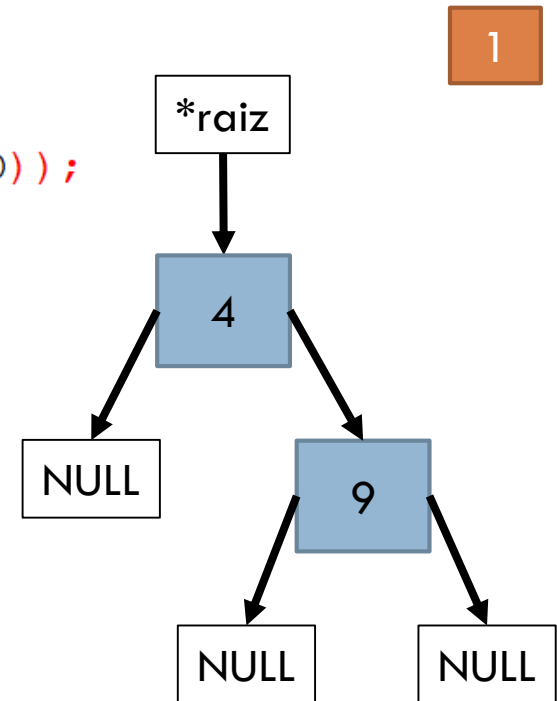
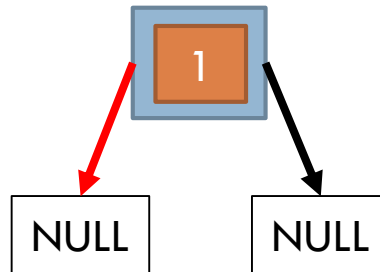
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    → novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

52

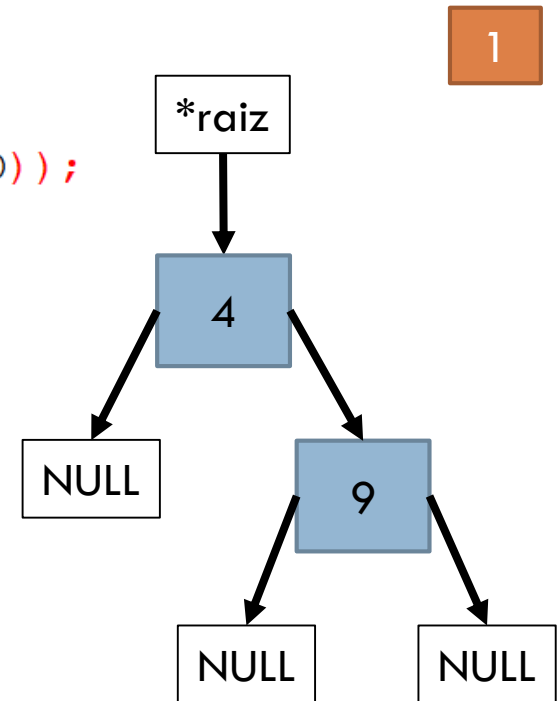
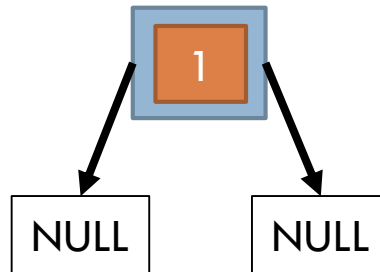
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    → novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

53

```
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;  
    [...]
```

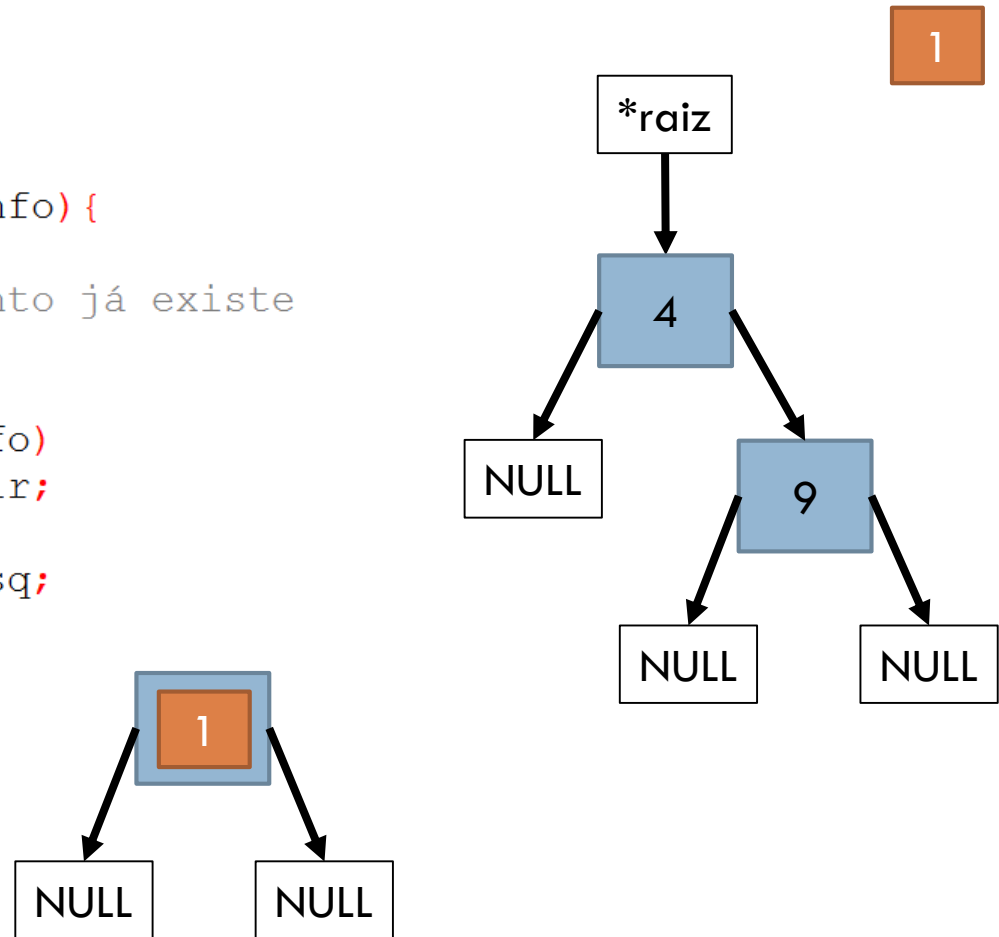


Inserção em ABB

54

```
→ 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
        }

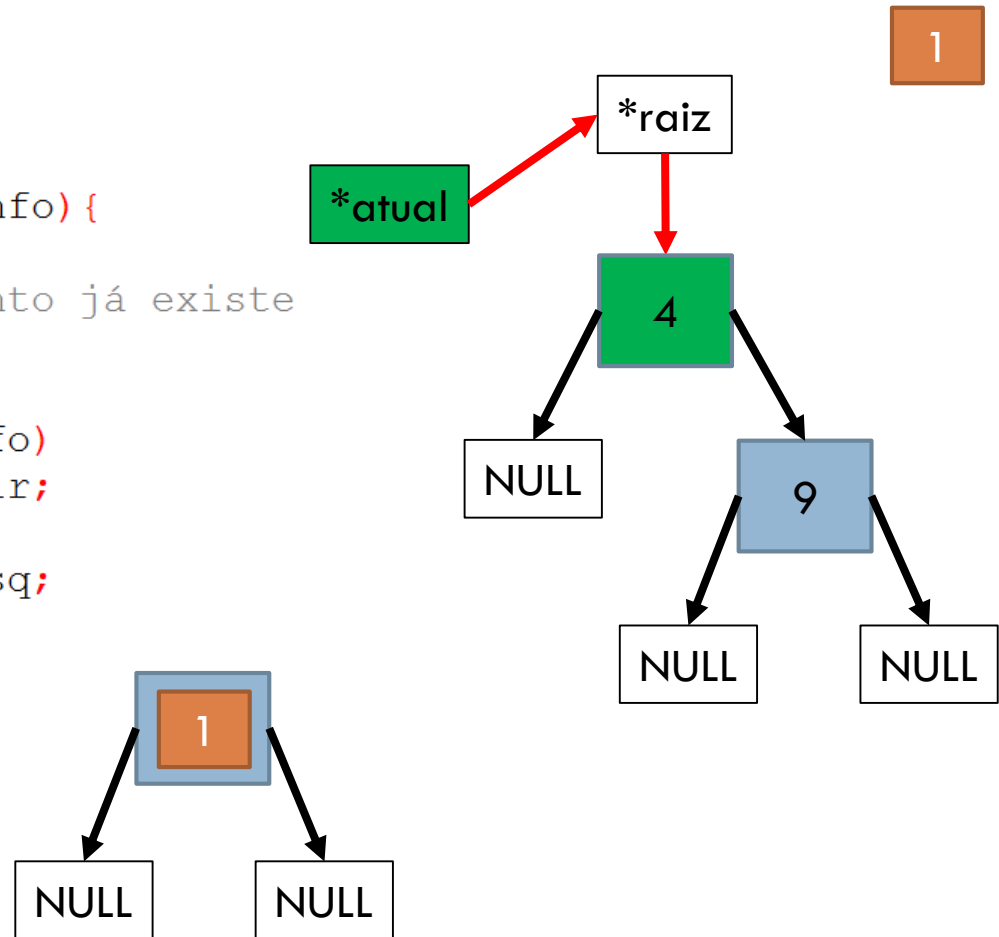
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

55

```
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;  
}
```

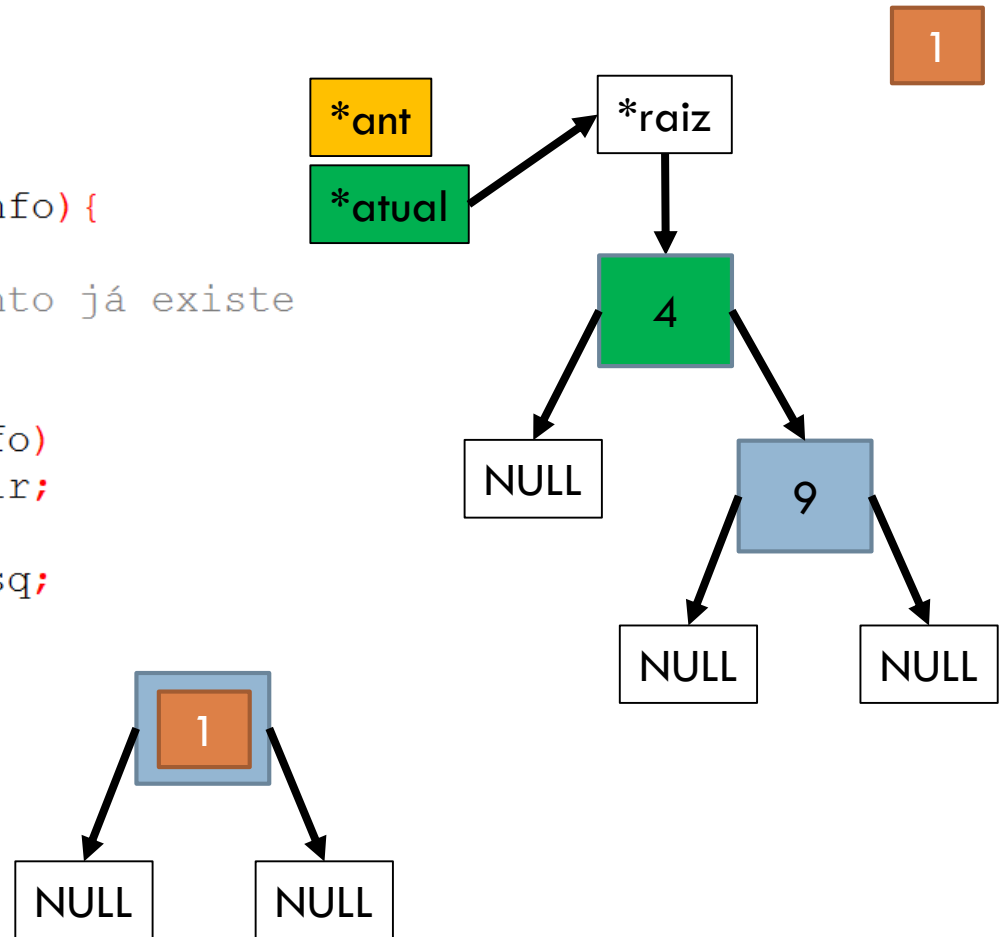


Inserção em ABB

56

```
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;
}
```

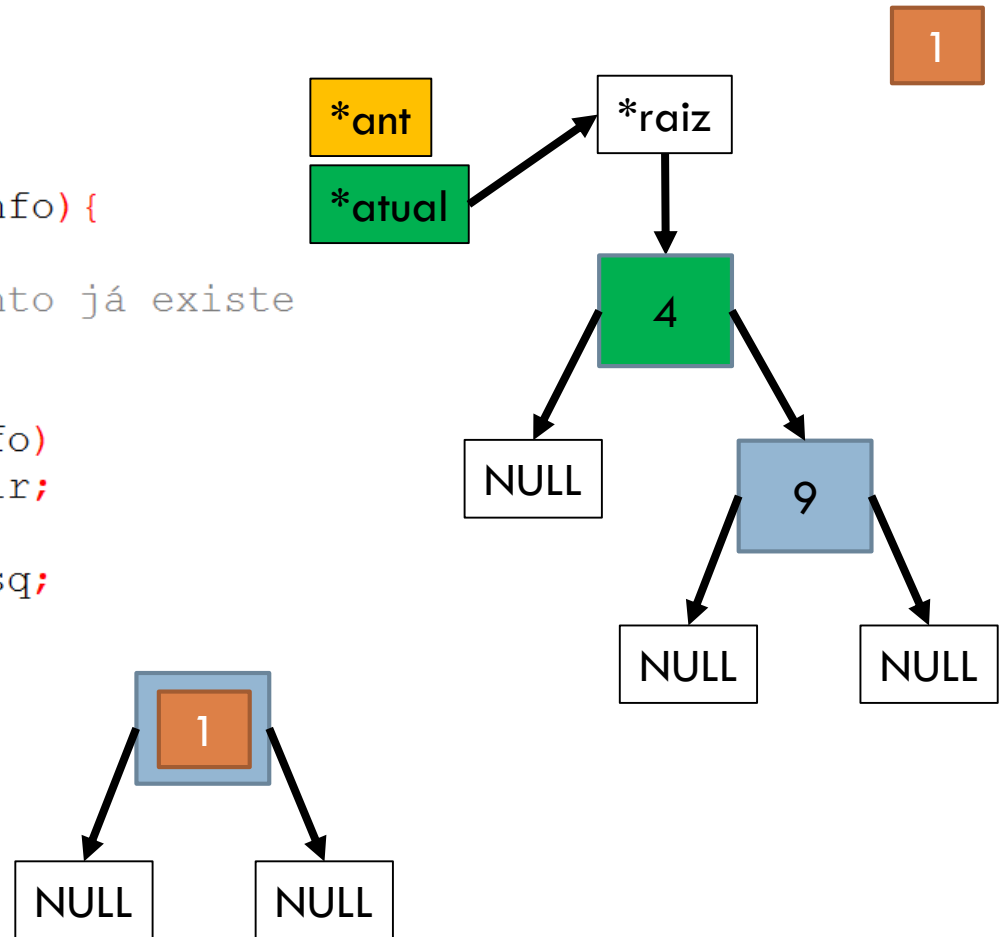


Inserção em ABB

57

```
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;
}
```

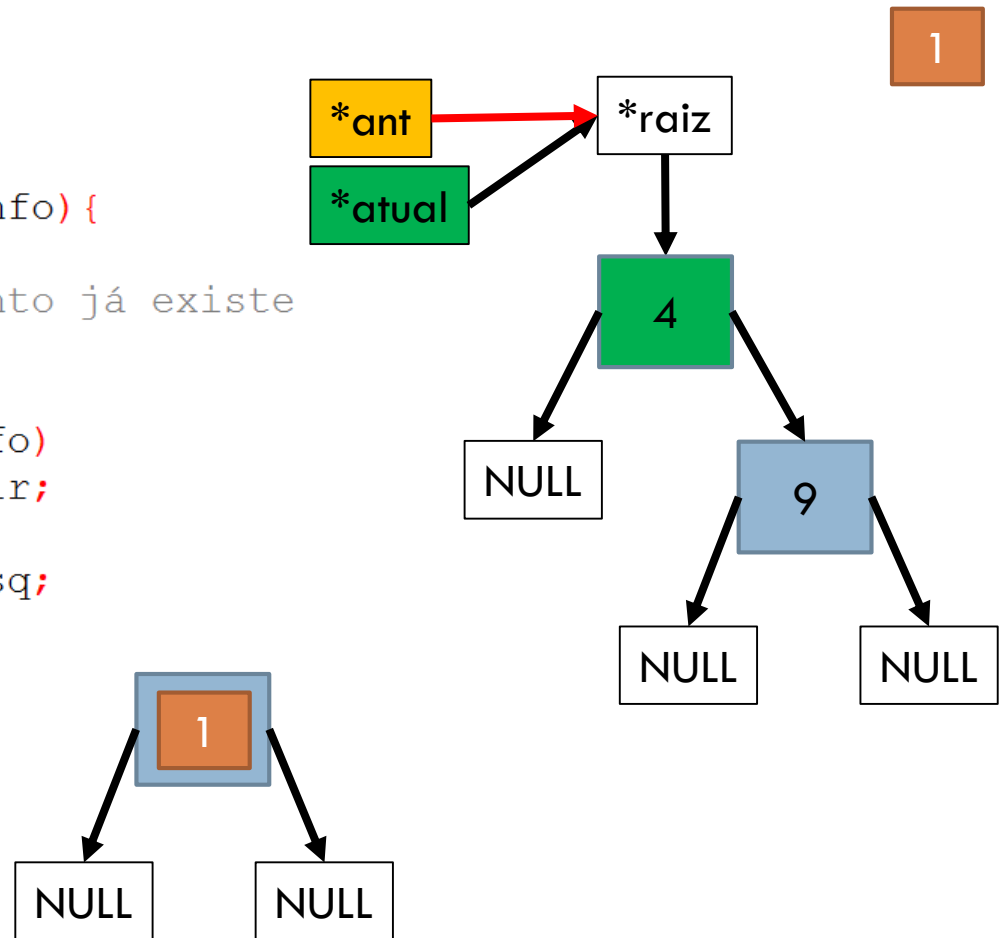


Inserção em ABB

58

```
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;
}
```

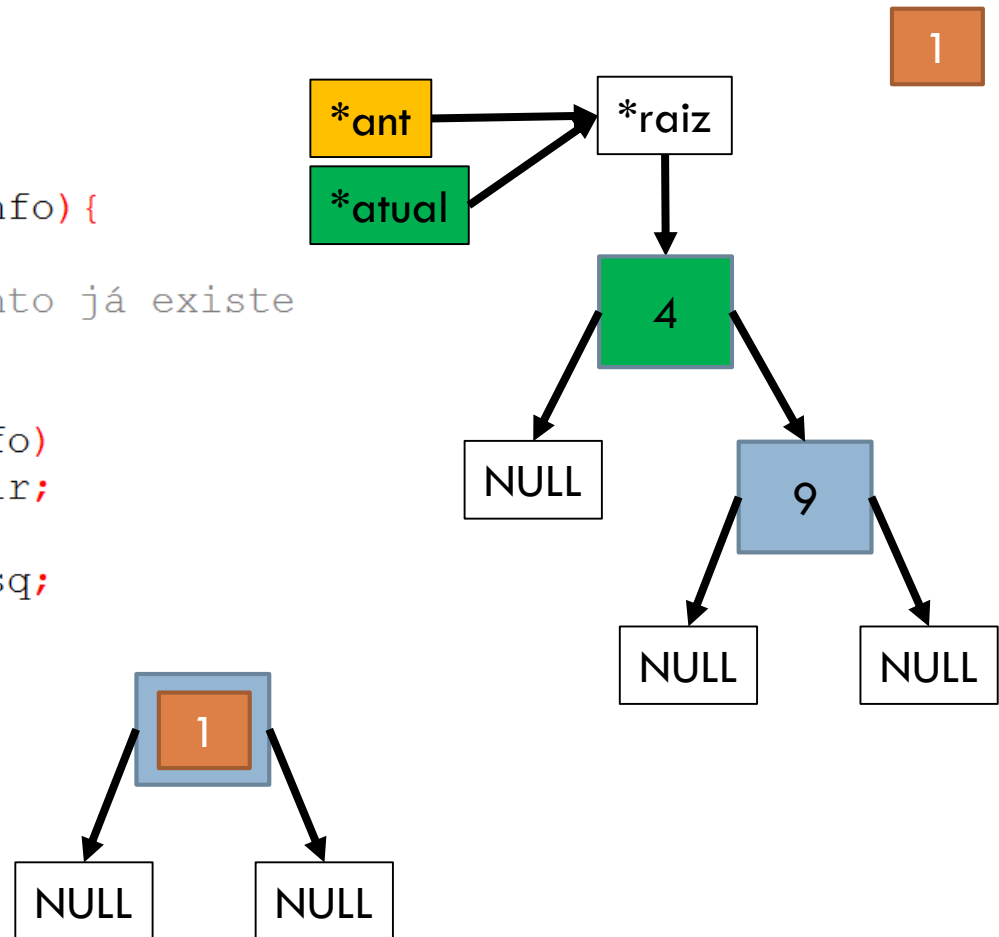


Inserção em ABB

59

```
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
        }

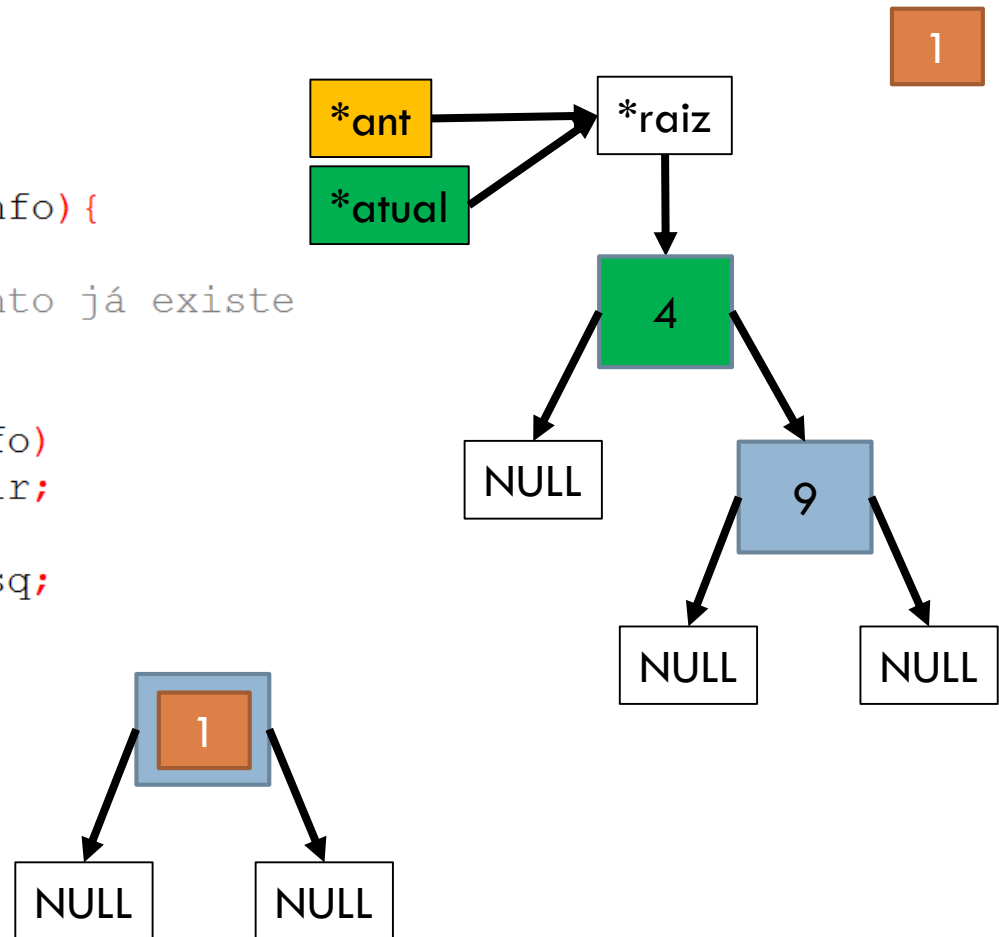
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

60

```
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;
}
```

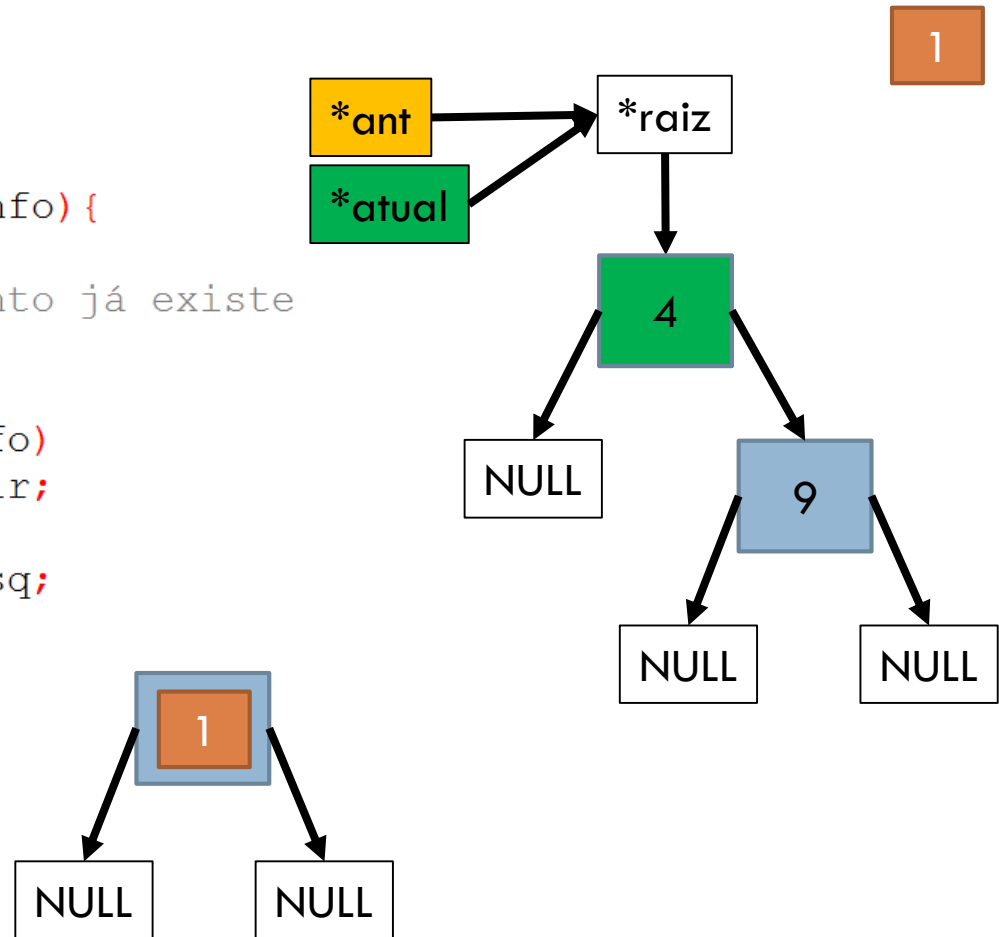


Inserção em ABB

61

```
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;
}
```

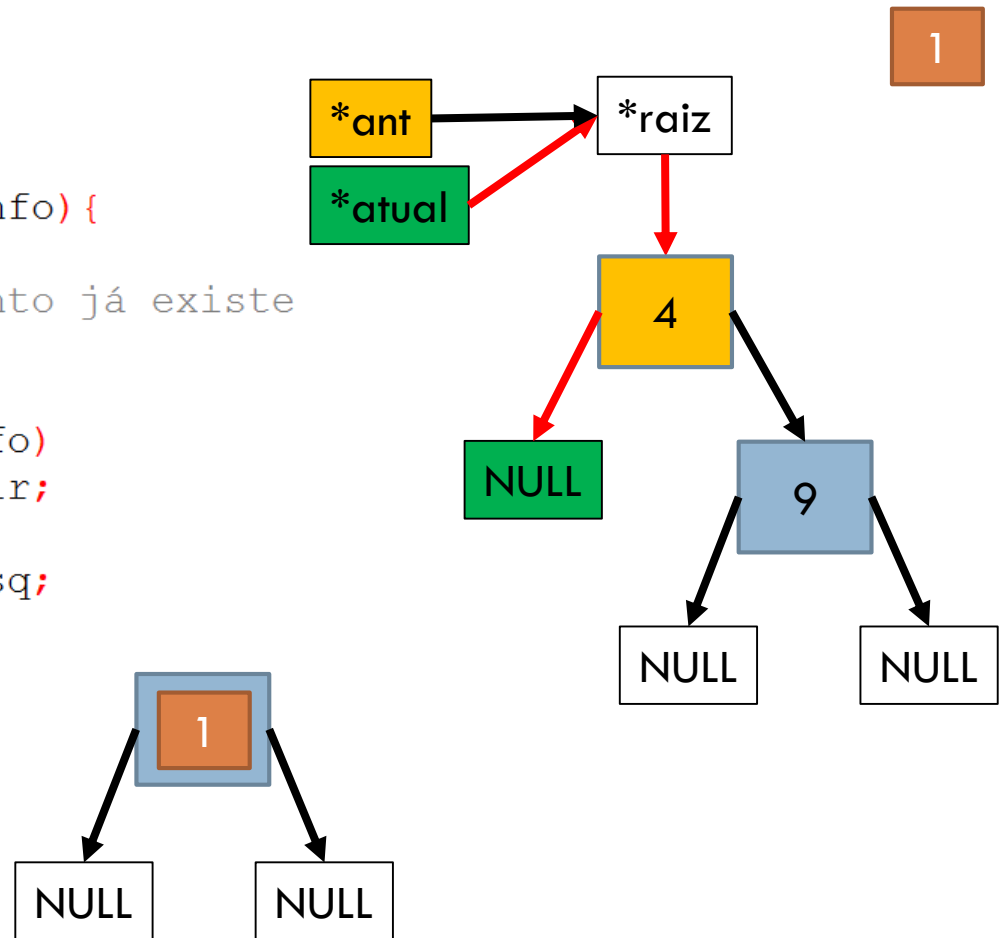


Inserção em ABB

62

```
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;
}
```

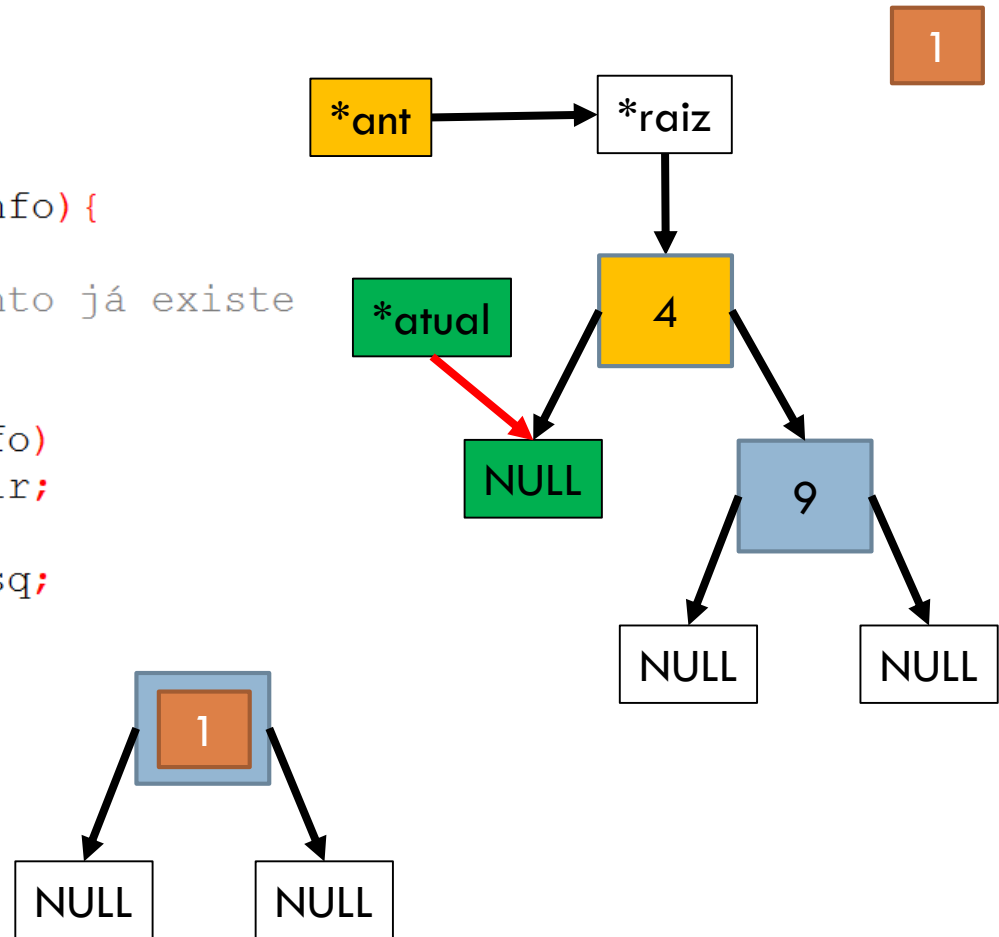


Inserção em ABB

63

```
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;
}
```

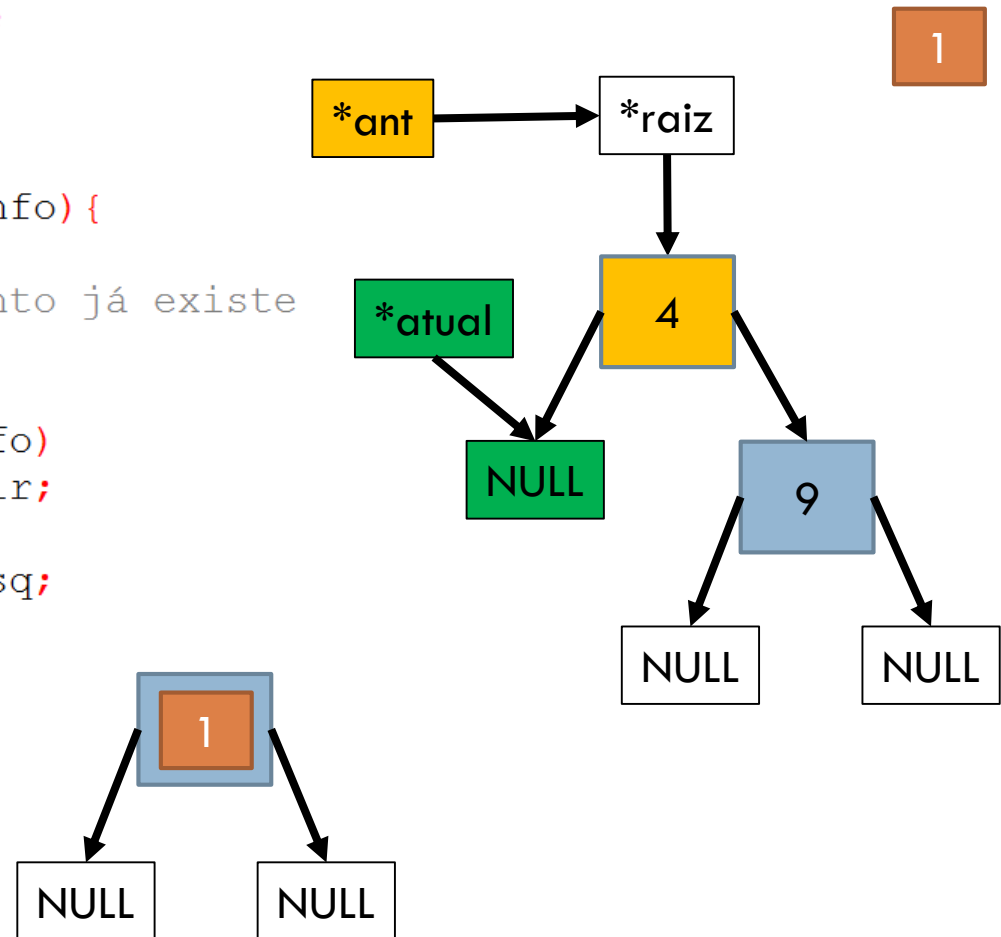


Inserção em ABB

64

```
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;
}
```

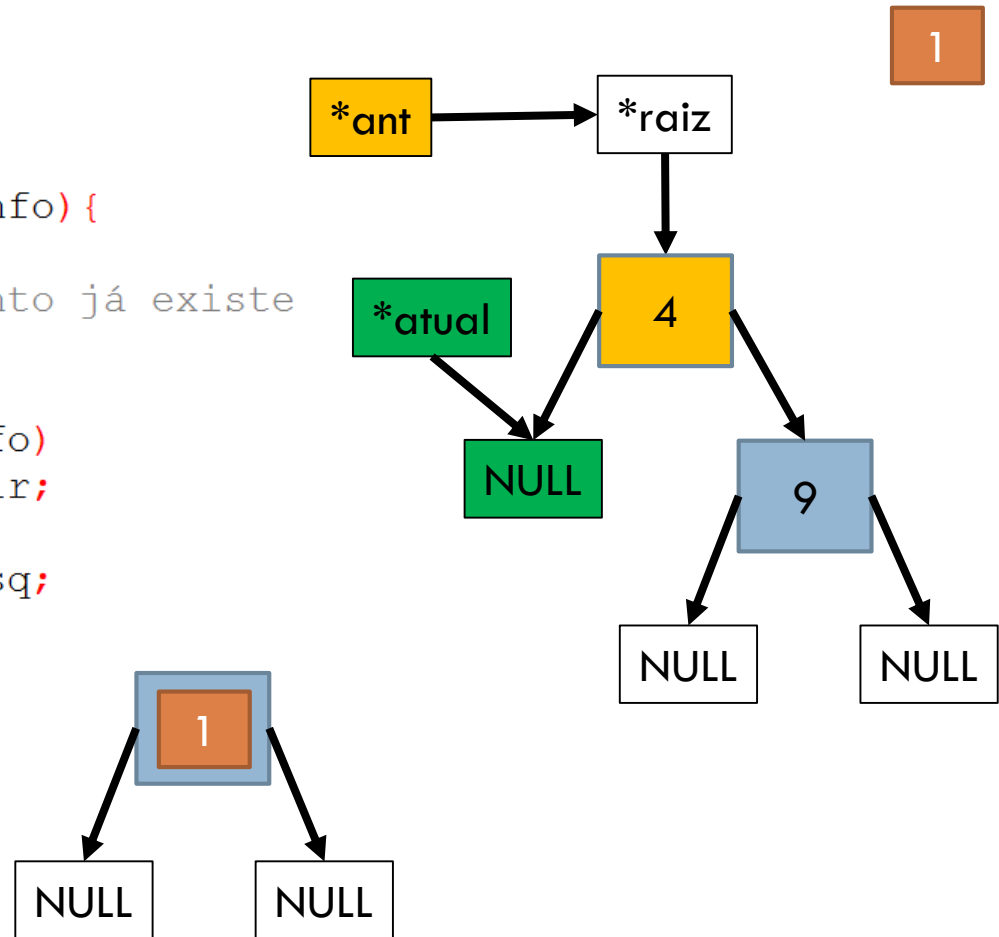


Inserção em ABB

65

```
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;
}
```

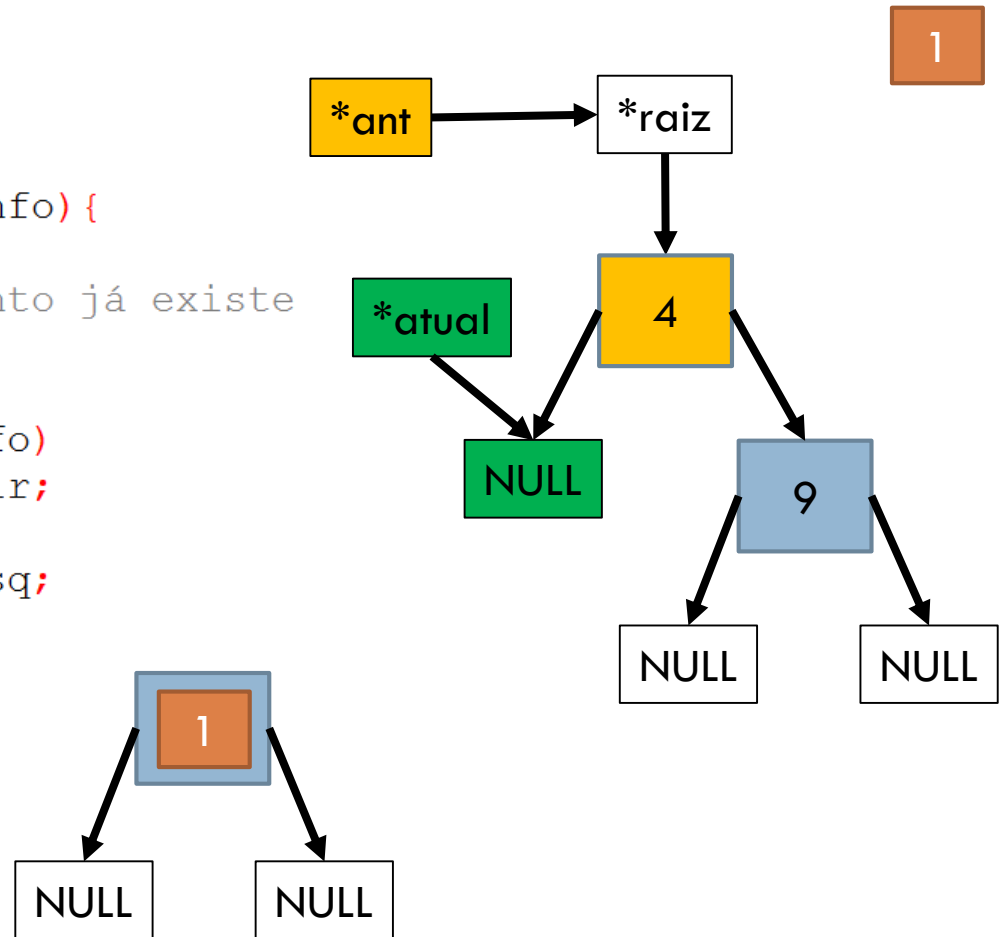


Inserção em ABB

66

```
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;
}
```

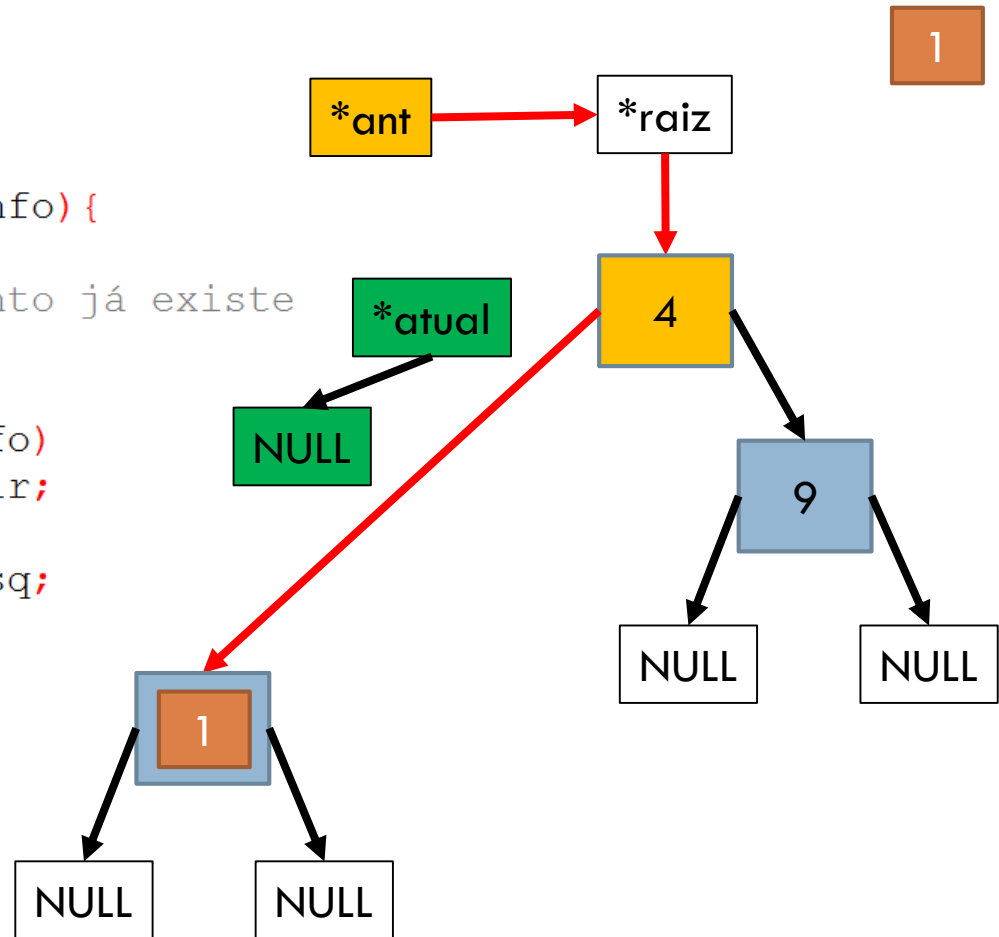


Inserção em ABB

67

```
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;
```

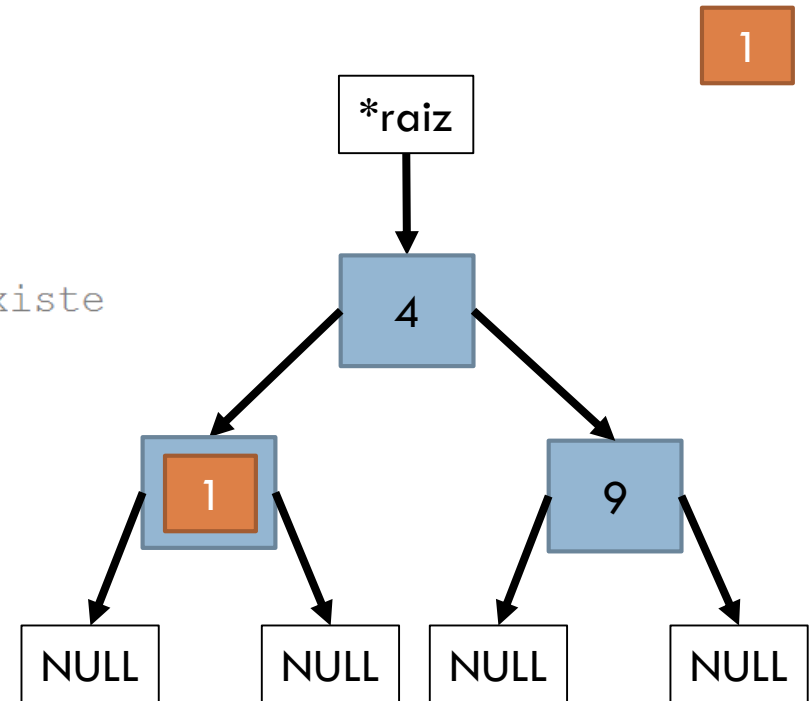


Inserção em ABB

68

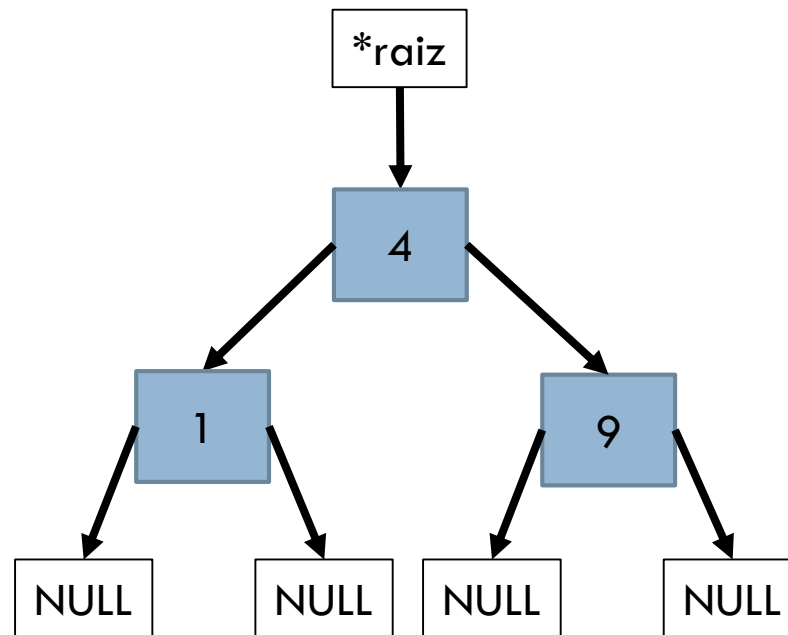
```
else{      [...]
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while(atual != NULL){
        ant = atual;
        if(valor == atual->info){
            free(novo);
            return 0; //elemento já existe
        }

        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

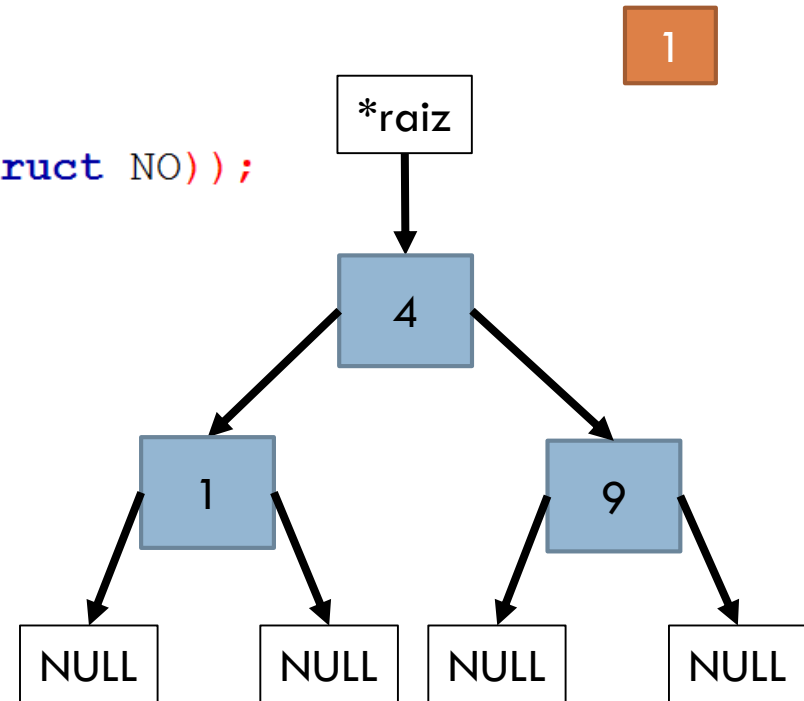
69



Inserção em ABB

70

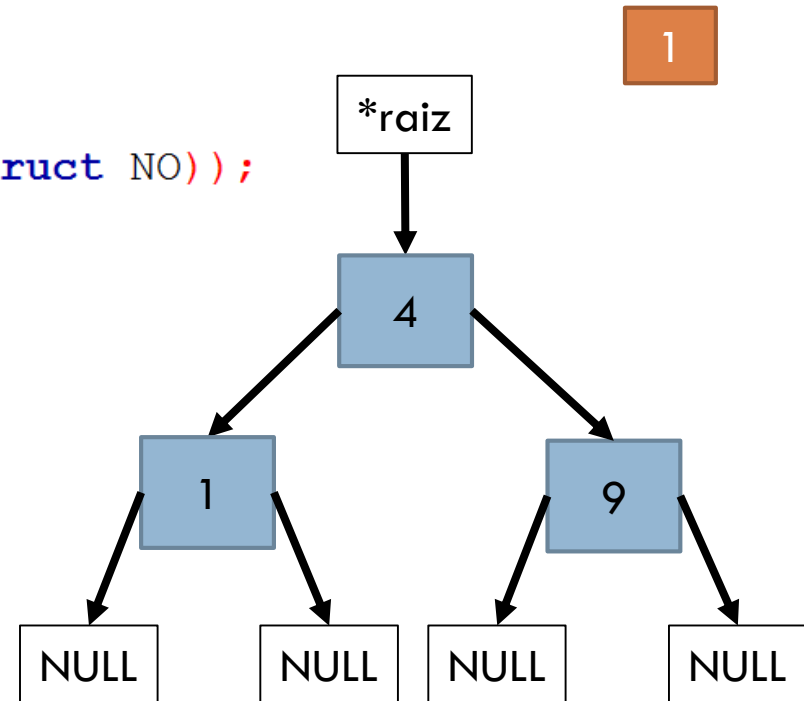
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

71

```
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;  
    [...]
```

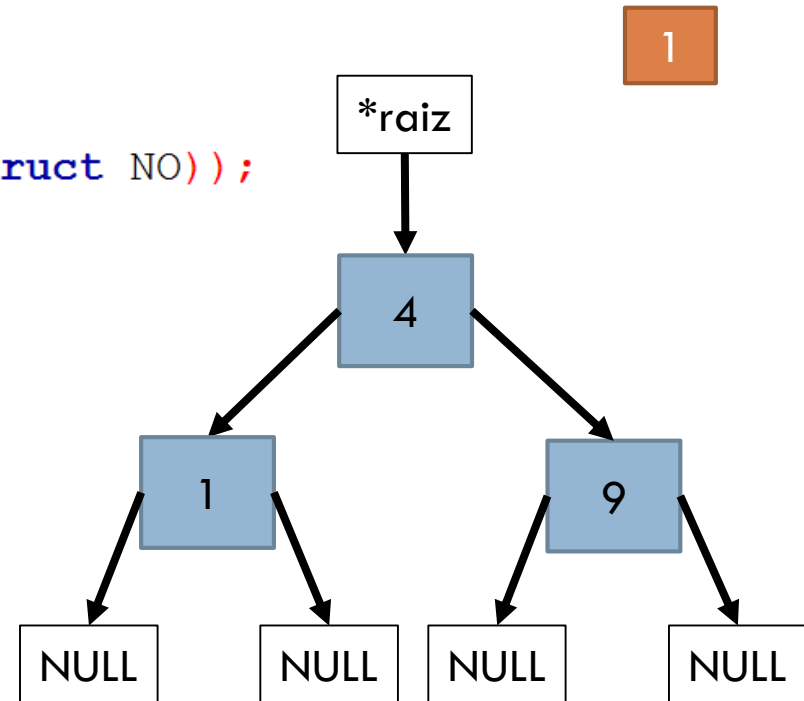


Inserção em ABB

72

```
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;  
      [...]
```

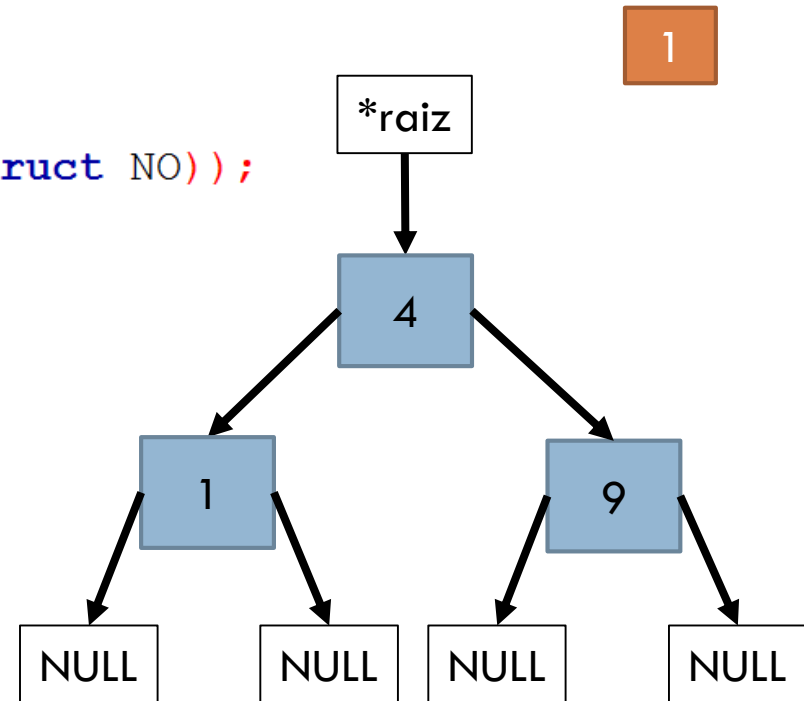
novo



Inserção em ABB

73

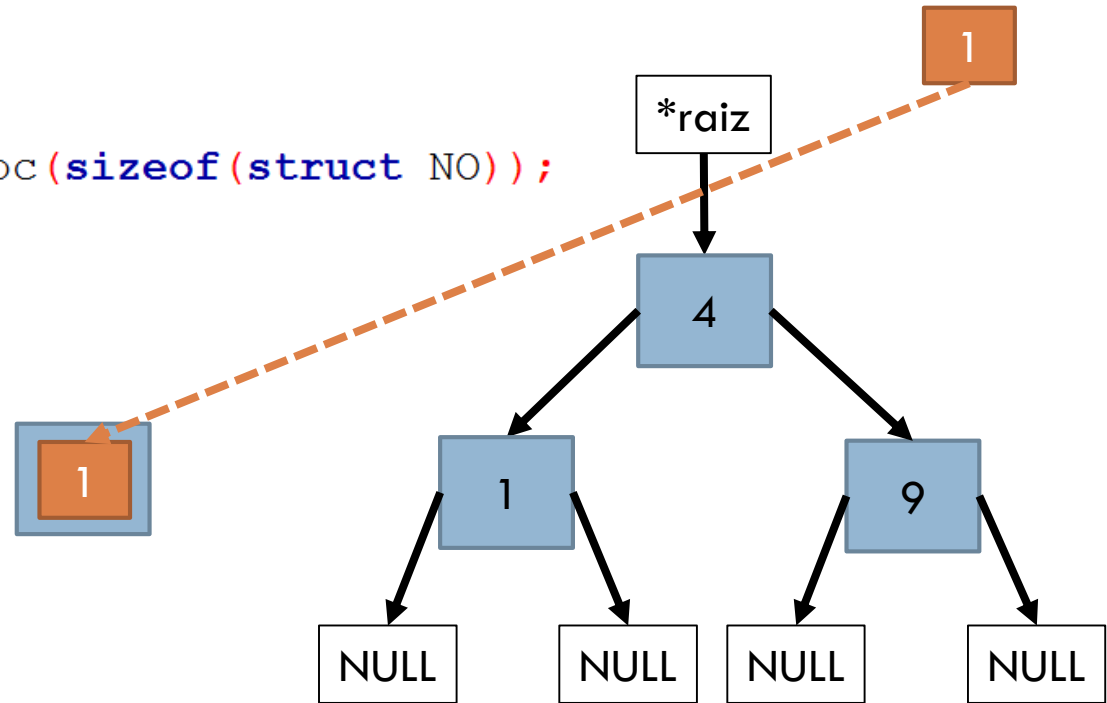
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    → if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

74

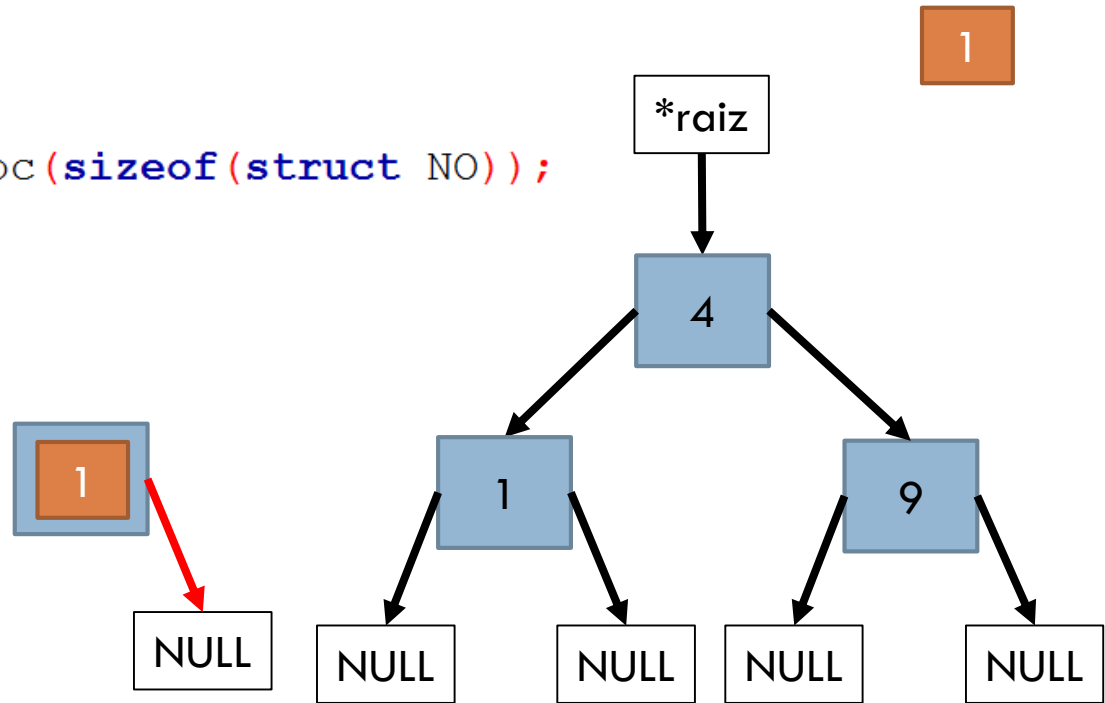
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    → novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

75

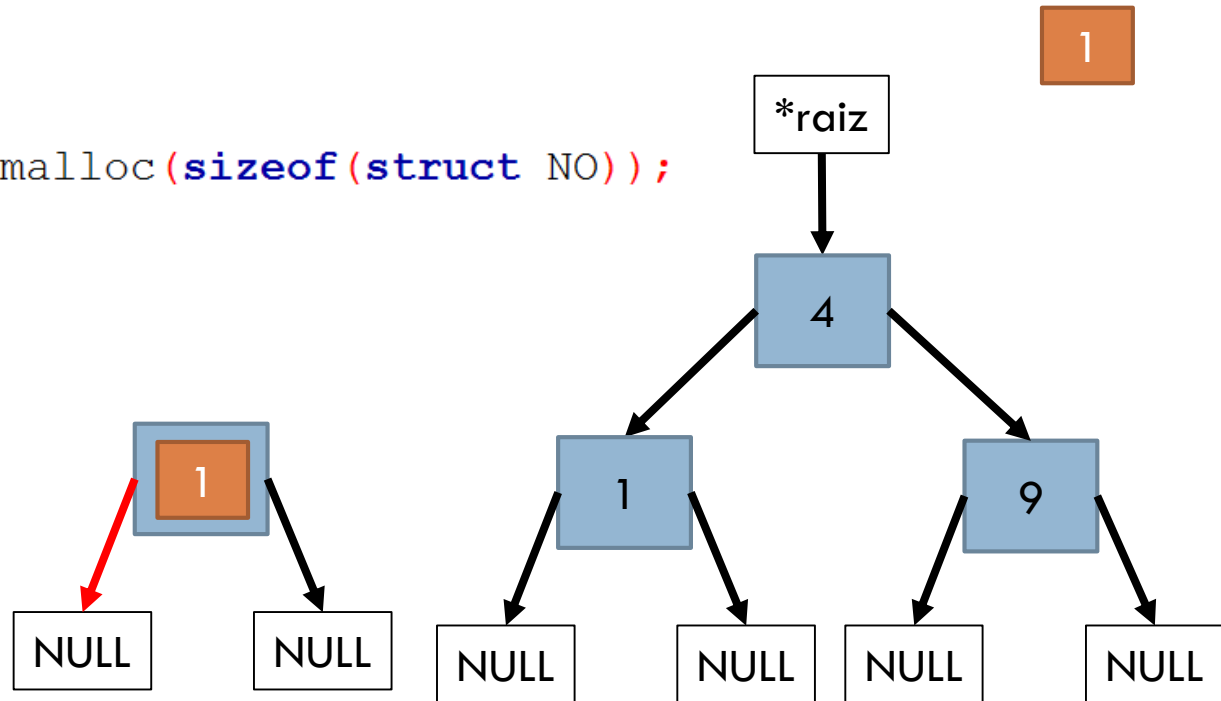
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    → novo->dir = NULL;  
    novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

76

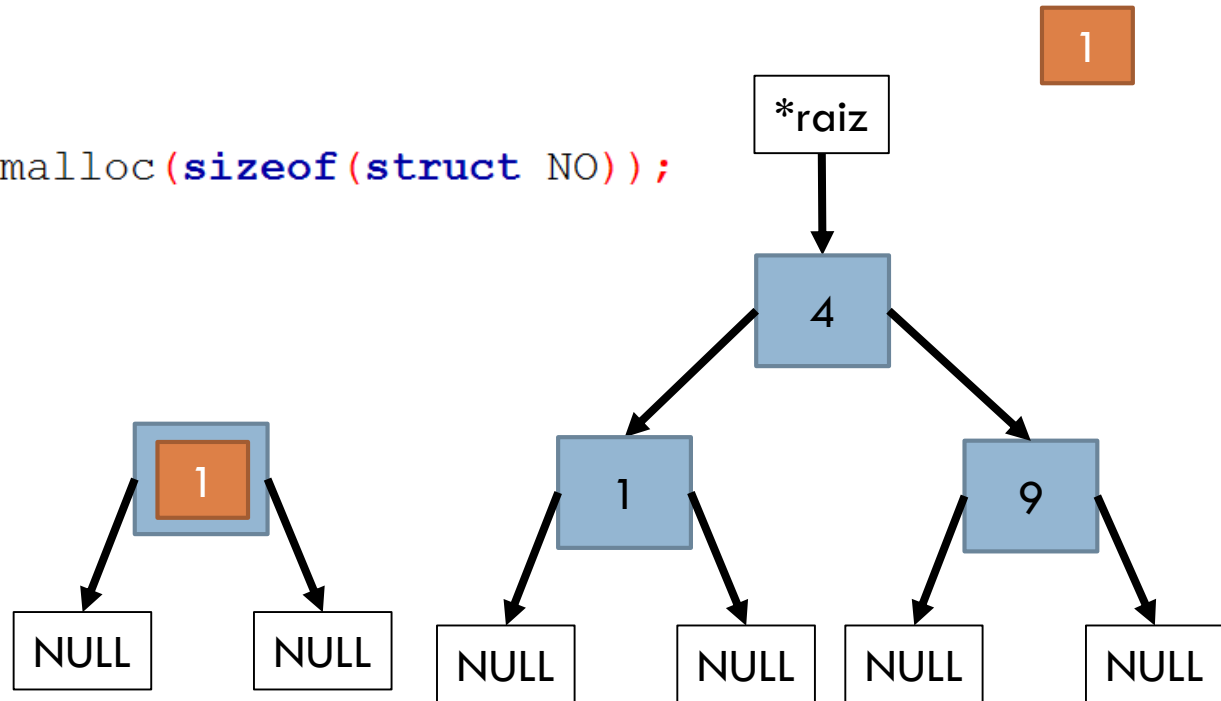
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL)  
        return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    → novo->esq = NULL;  
  
    if(*raiz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



Inserção em ABB

77

```
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;  
    [...]
```

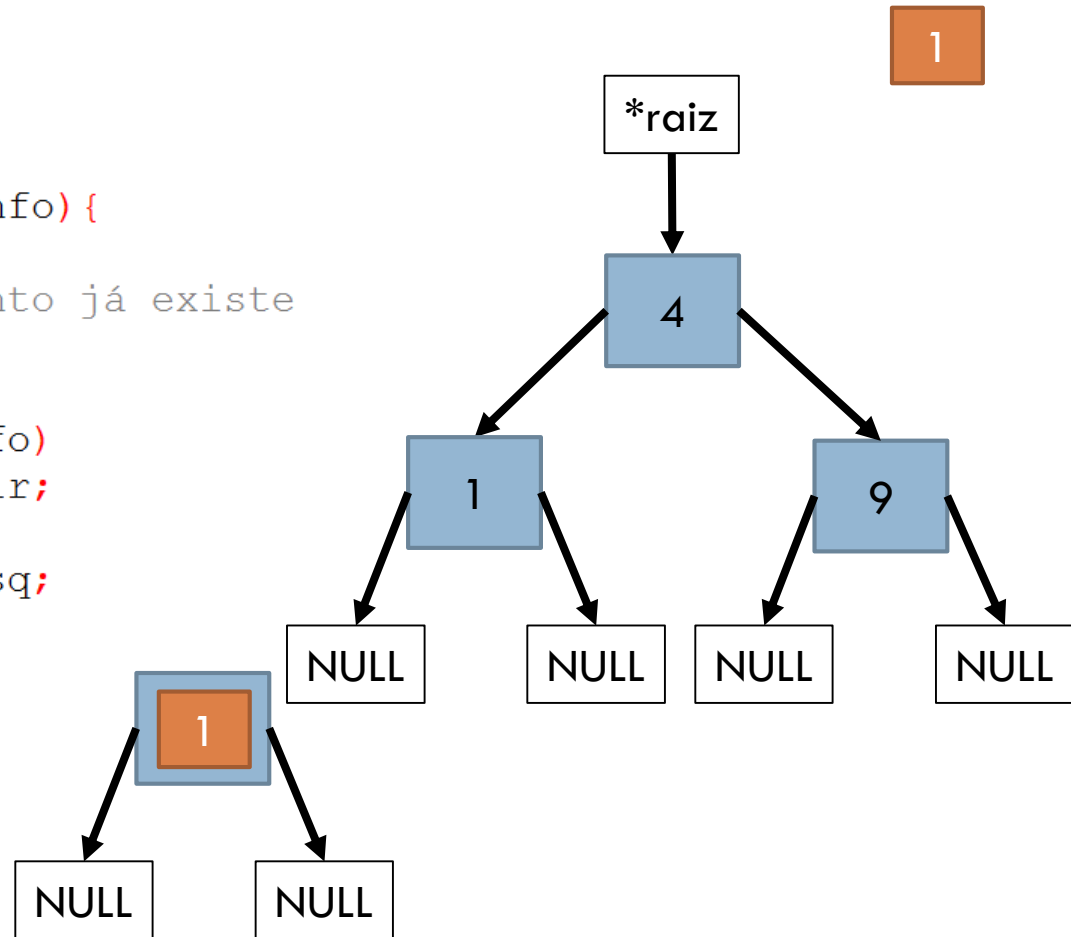


Inserção em ABB

78

```
→ 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
        }

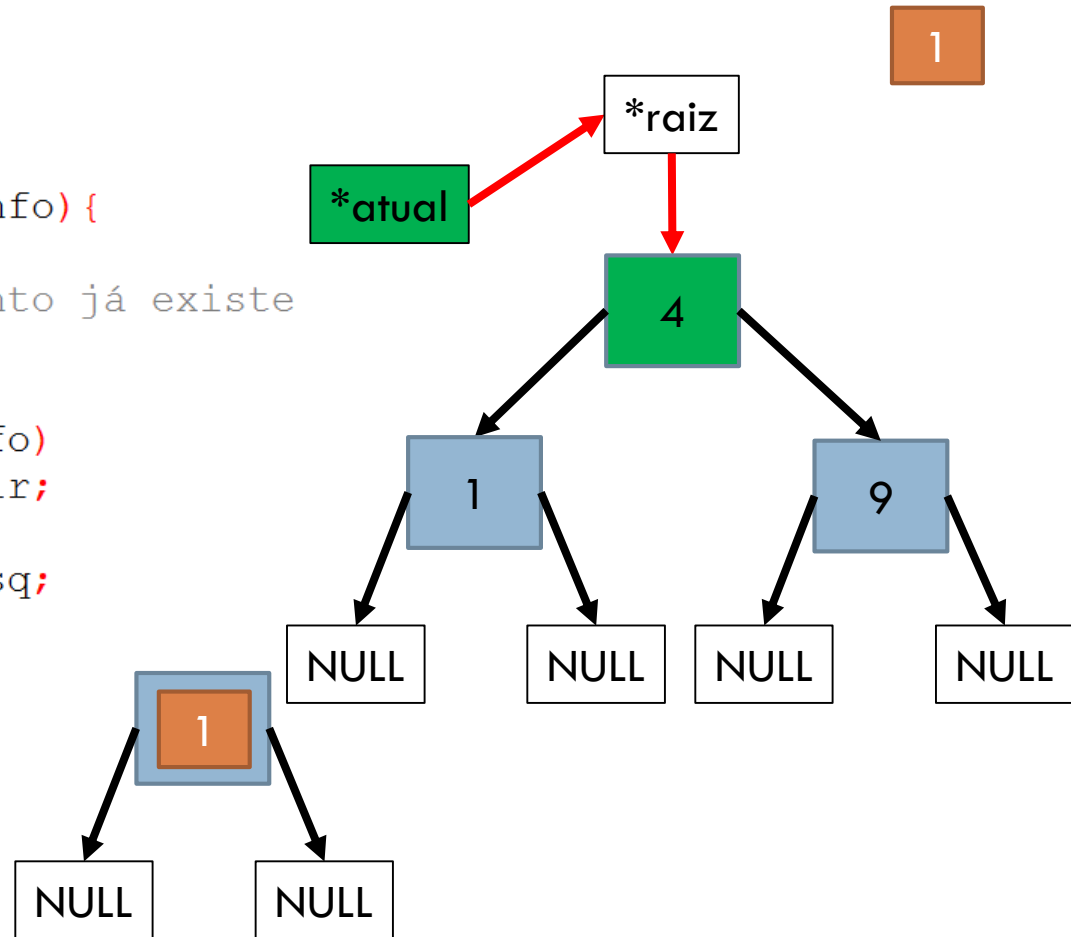
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

79

```
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;  
}
```

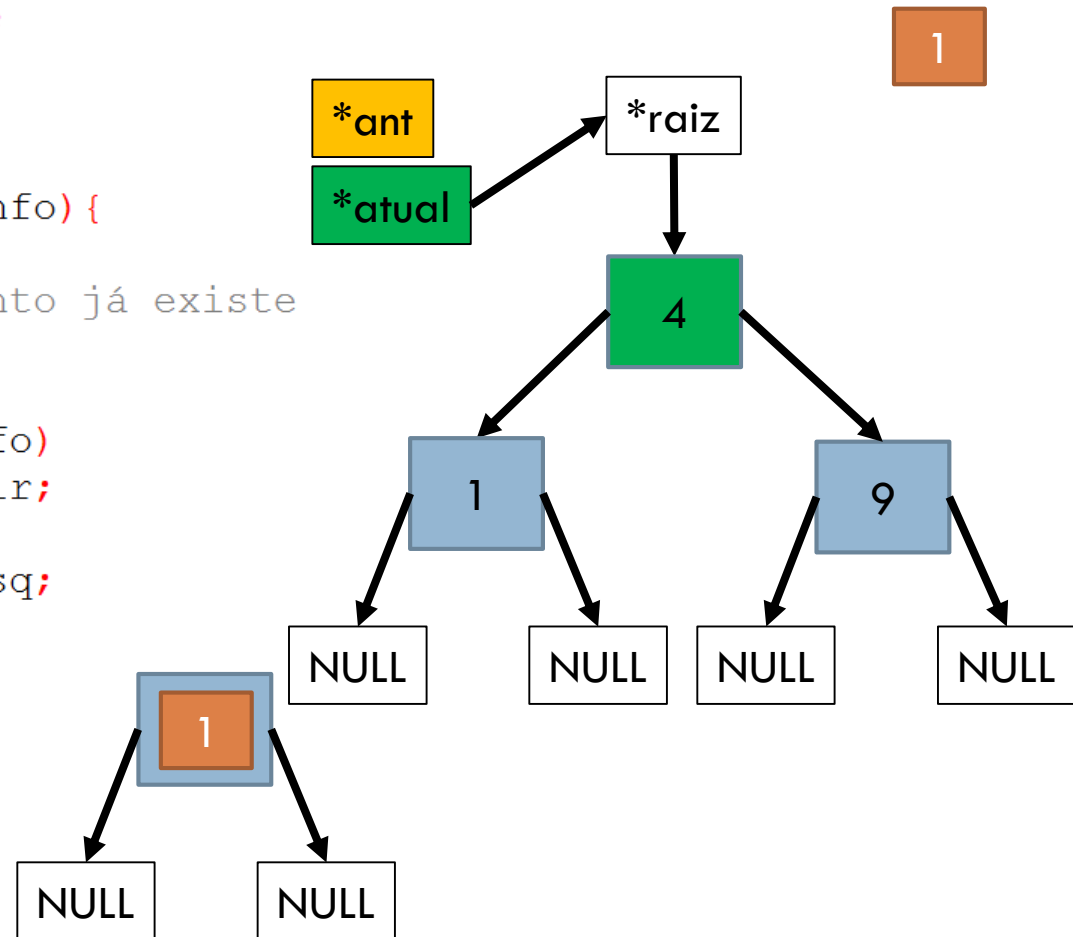


Inserção em ABB

80

```
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;
}
```

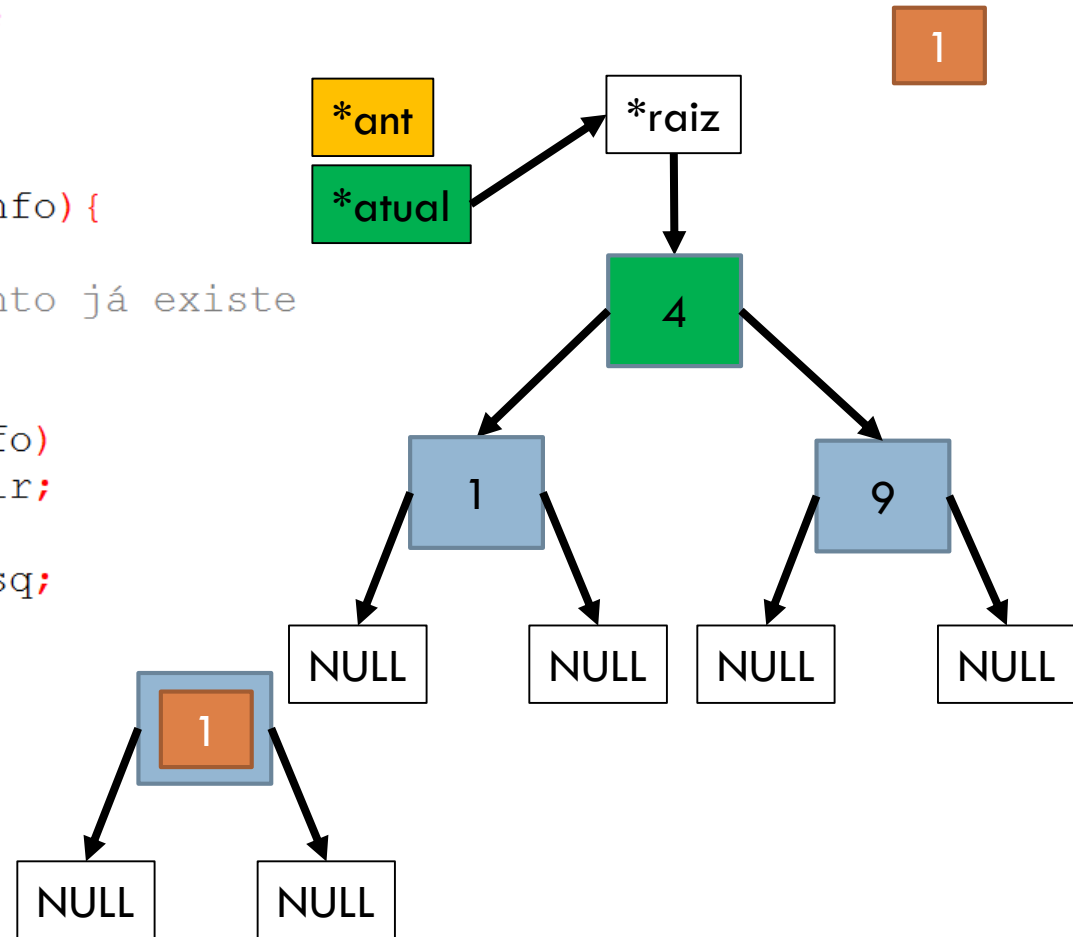


Inserção em ABB

81

```
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;
}
```

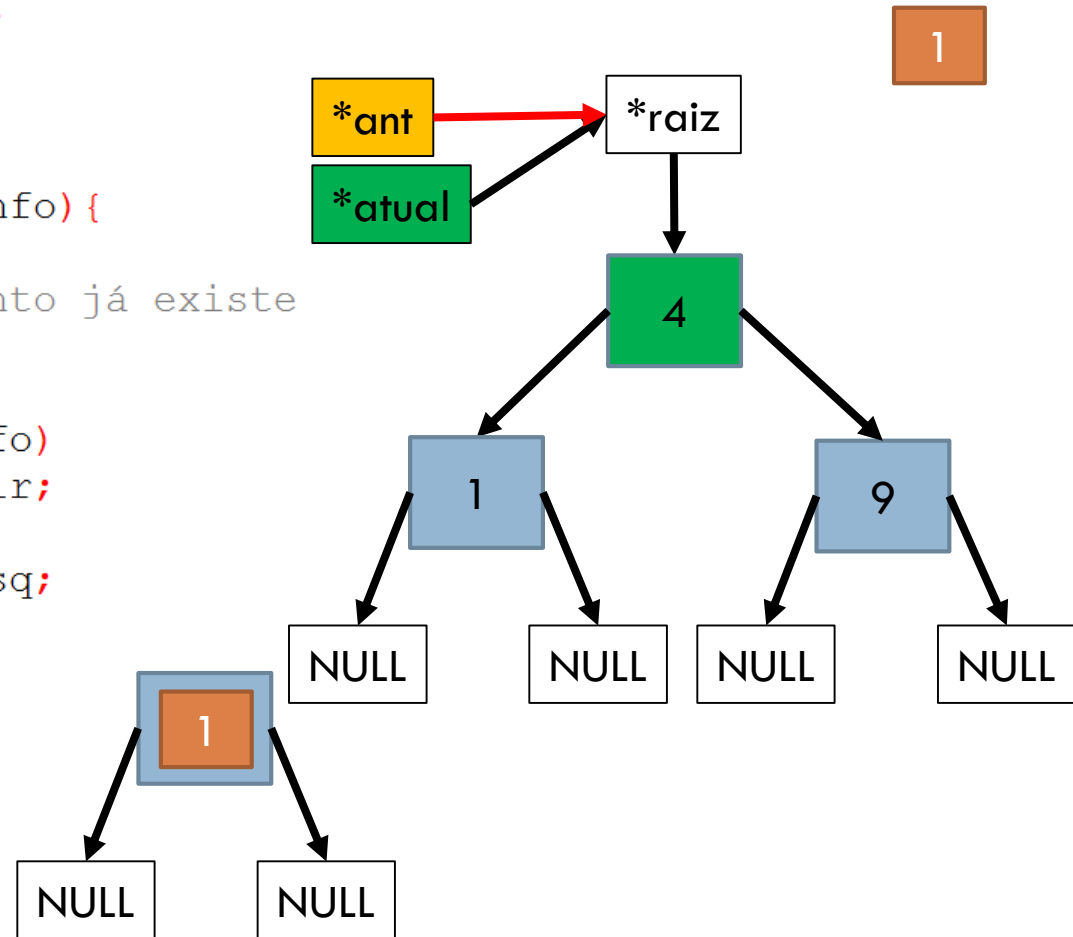


Inserção em ABB

82

```
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;
}
```

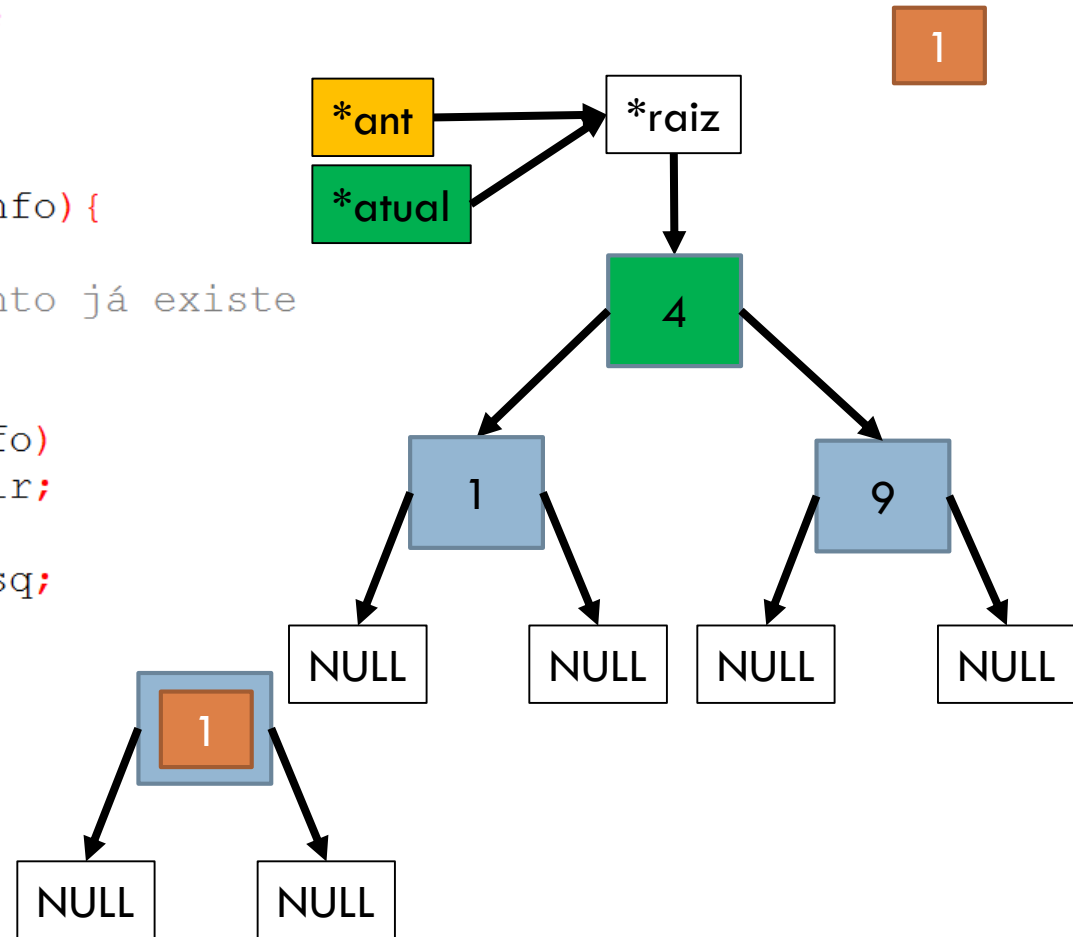


Inserção em ABB

83

```
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
        }

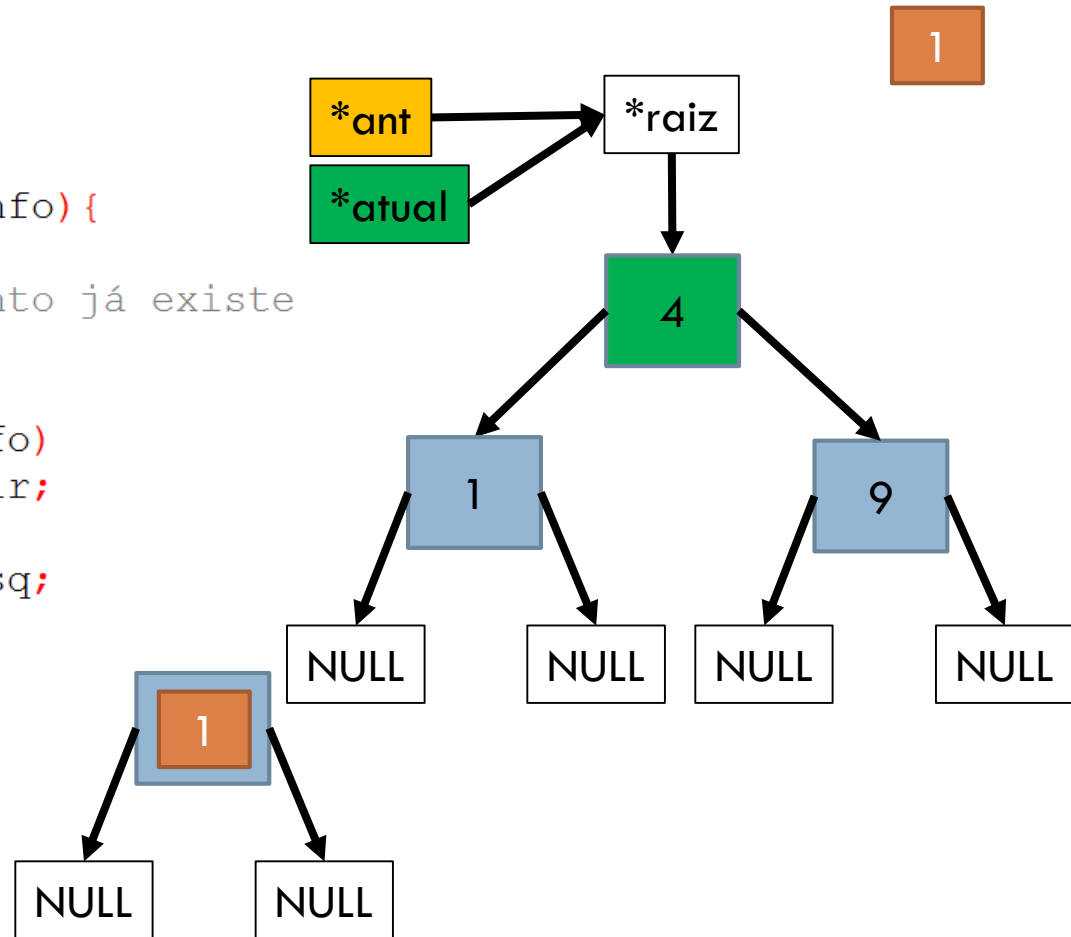
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

84

```
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;
}
```

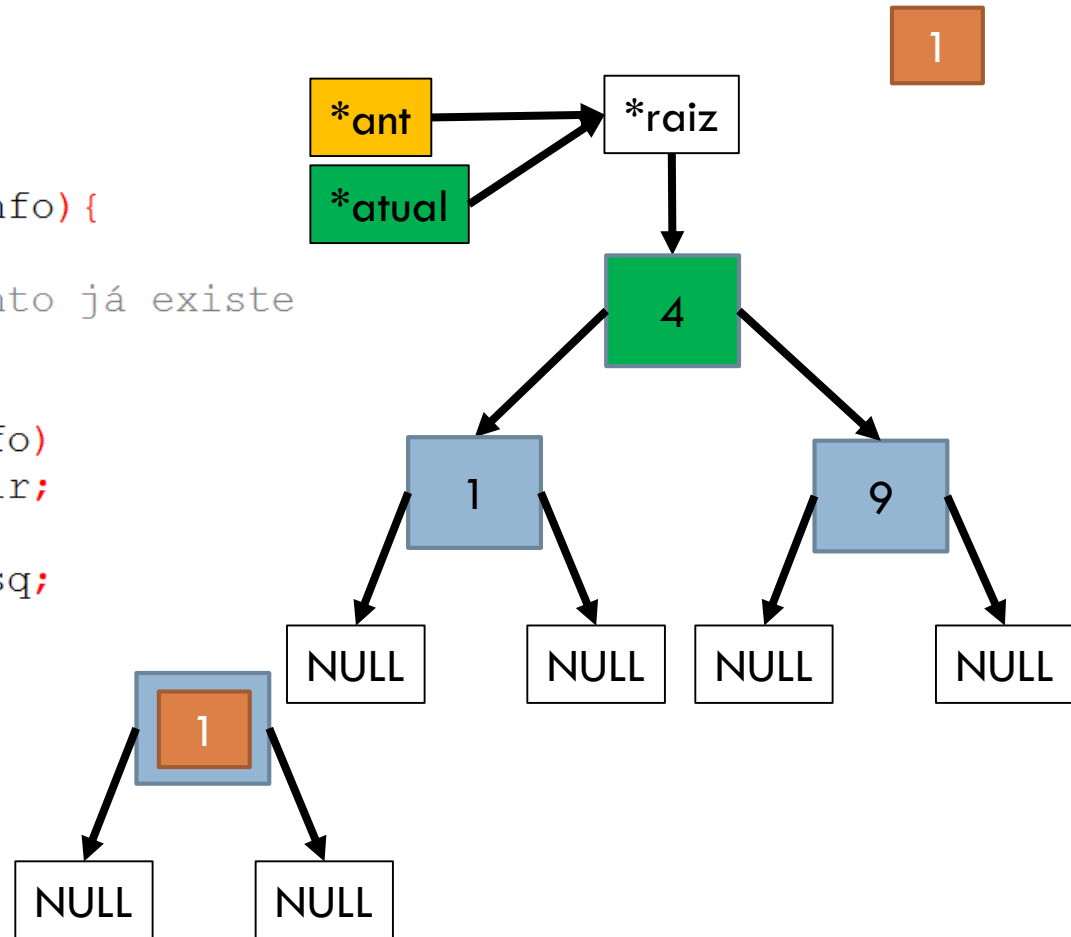


Inserção em ABB

85

```
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;
}
```

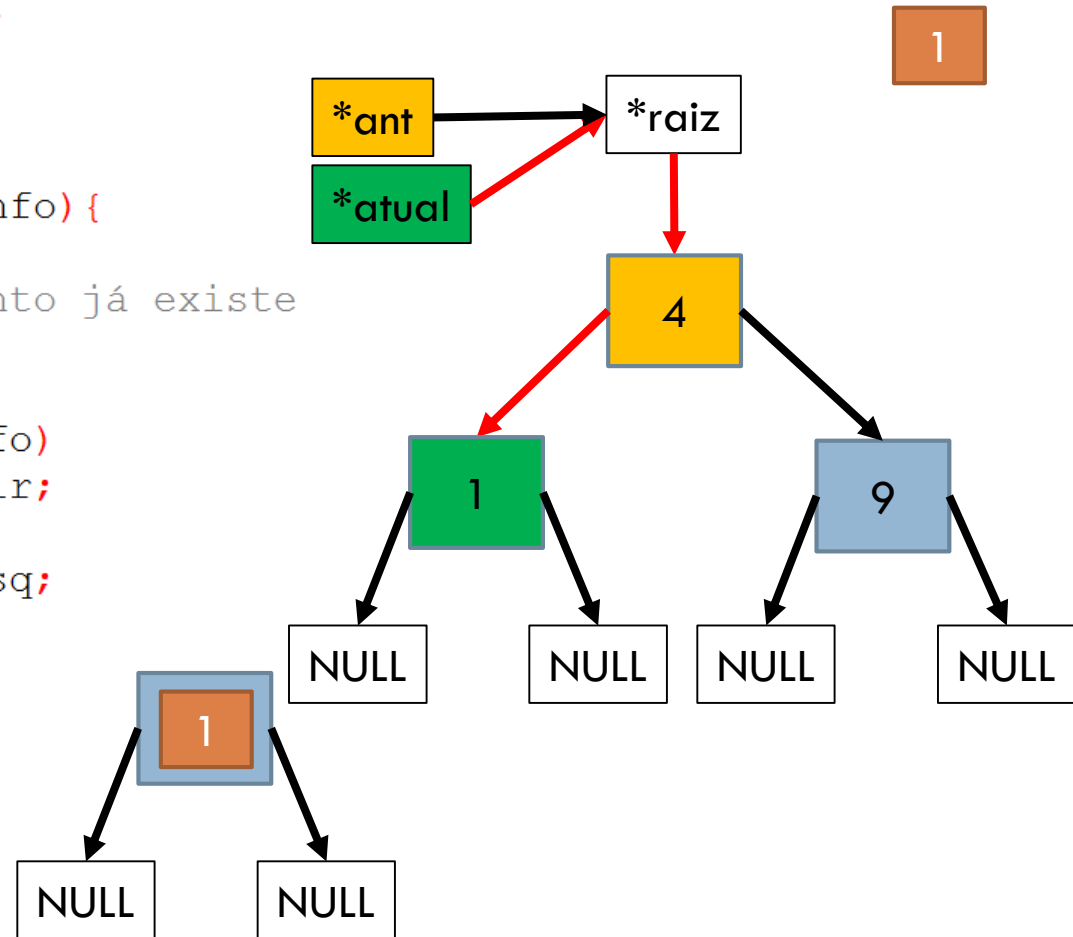


Inserção em ABB

86

```
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
        }

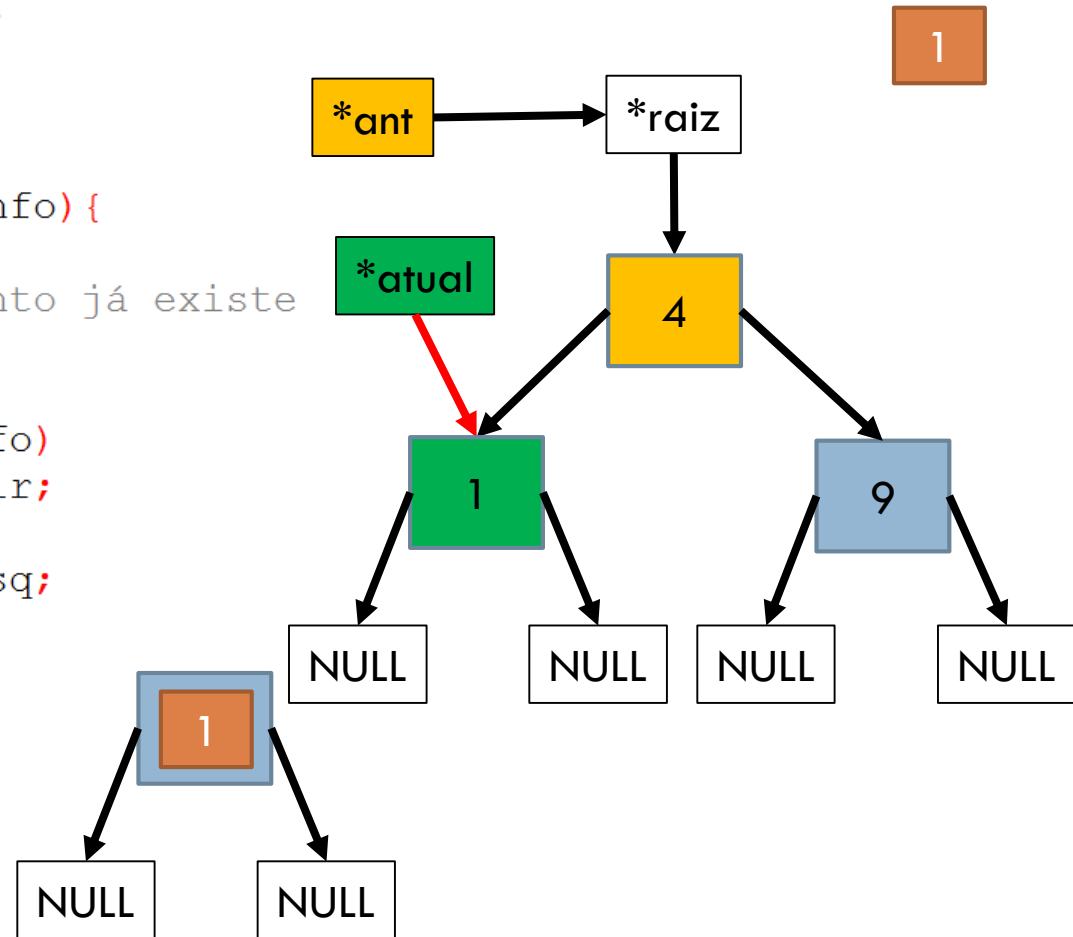
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

87

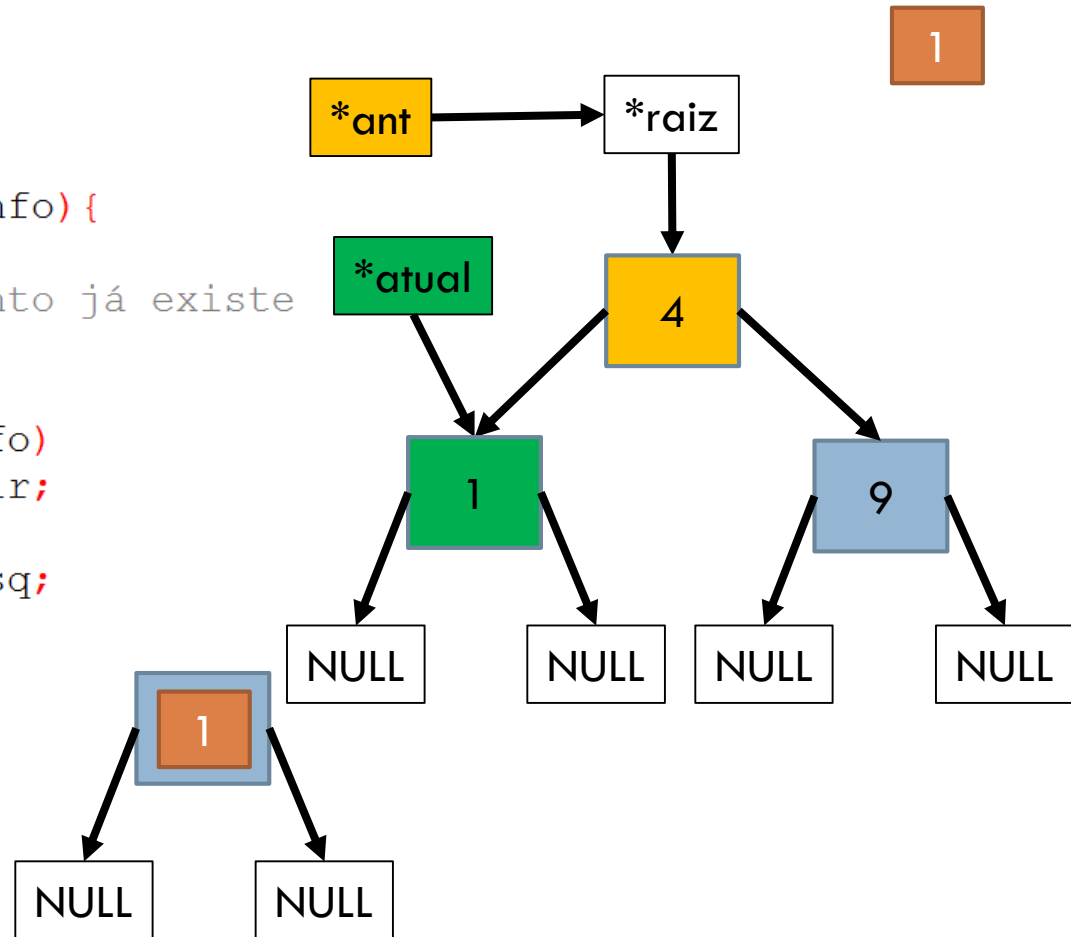
```
else{  
    [...]  
    struct NO* atual = *raiz;  
    struct NO* ant = NULL;  
    while(atual != NULL){  
        ant = atual;  
        if(valor == atual->info){  
            free(novo);  
            return 0; //elemento já existe  
        }  
        if(valor > atual->info){  
            atual = atual->dir;  
        }  
        else  
            atual = atual->esq;  
    }  
    if(valor > ant->info)  
        ant->dir = novo;  
    else  
        ant->esq = novo;  
}  
return 1;  
}
```



Inserção em ABB

88

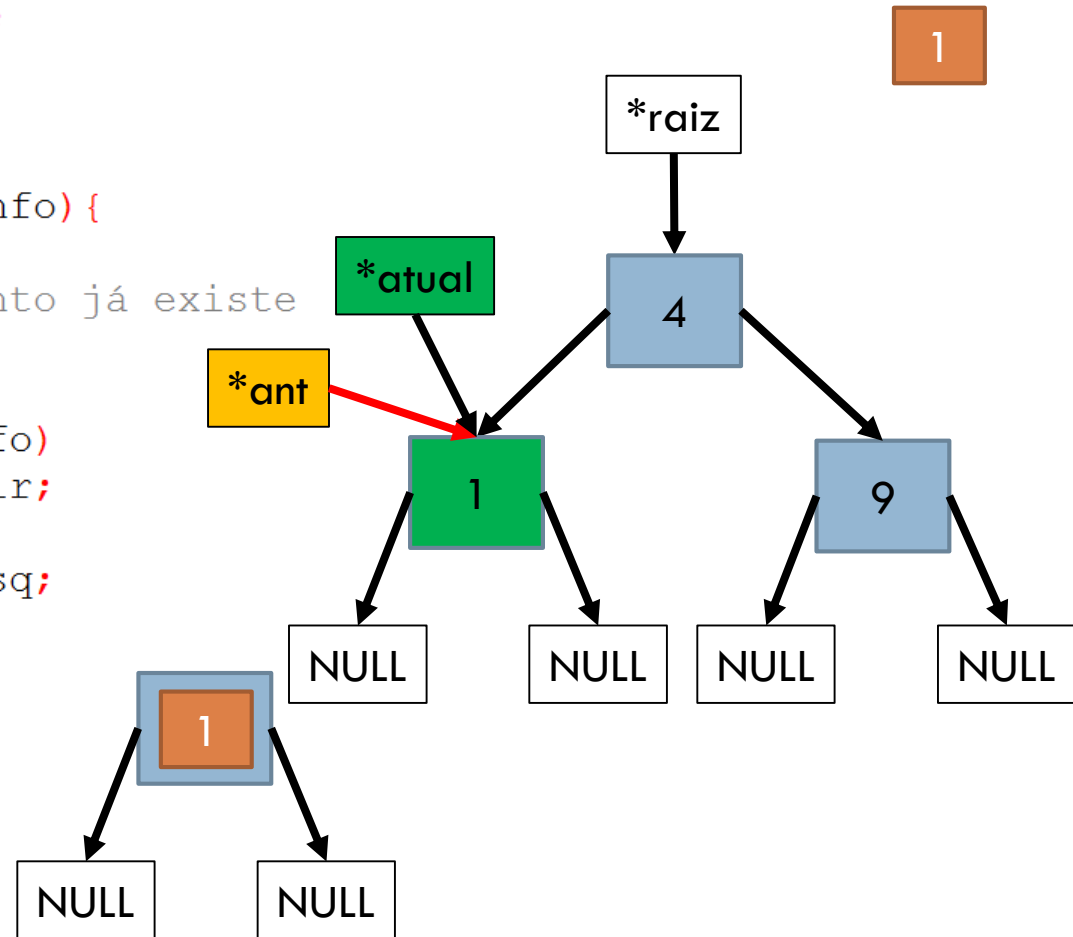
```
else{  
    [...]  
    struct NO* atual = *raiz;  
    struct NO* ant = NULL;  
    → while(atual != NULL){  
        ant = atual;  
        if(valor == atual->info){  
            free(novo);  
            return 0; //elemento já existe  
        }  
  
        if(valor > atual->info)  
            atual = atual->dir;  
        else  
            atual = atual->esq;  
    }  
    if(valor > ant->info)  
        ant->dir = novo;  
    else  
        ant->esq = novo;  
}  
return 1;  
}
```



Inserção em ABB

89

```
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;
```

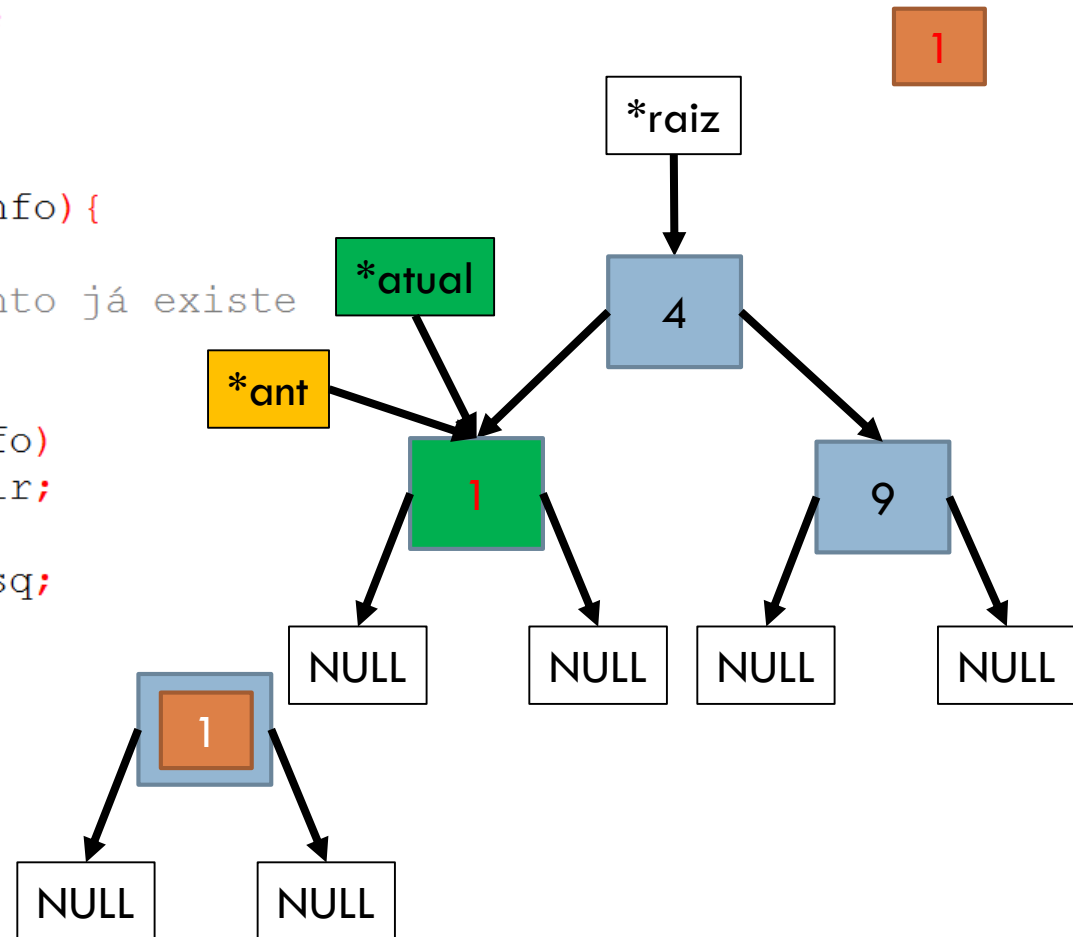


Inserção em ABB

90

```
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
        }

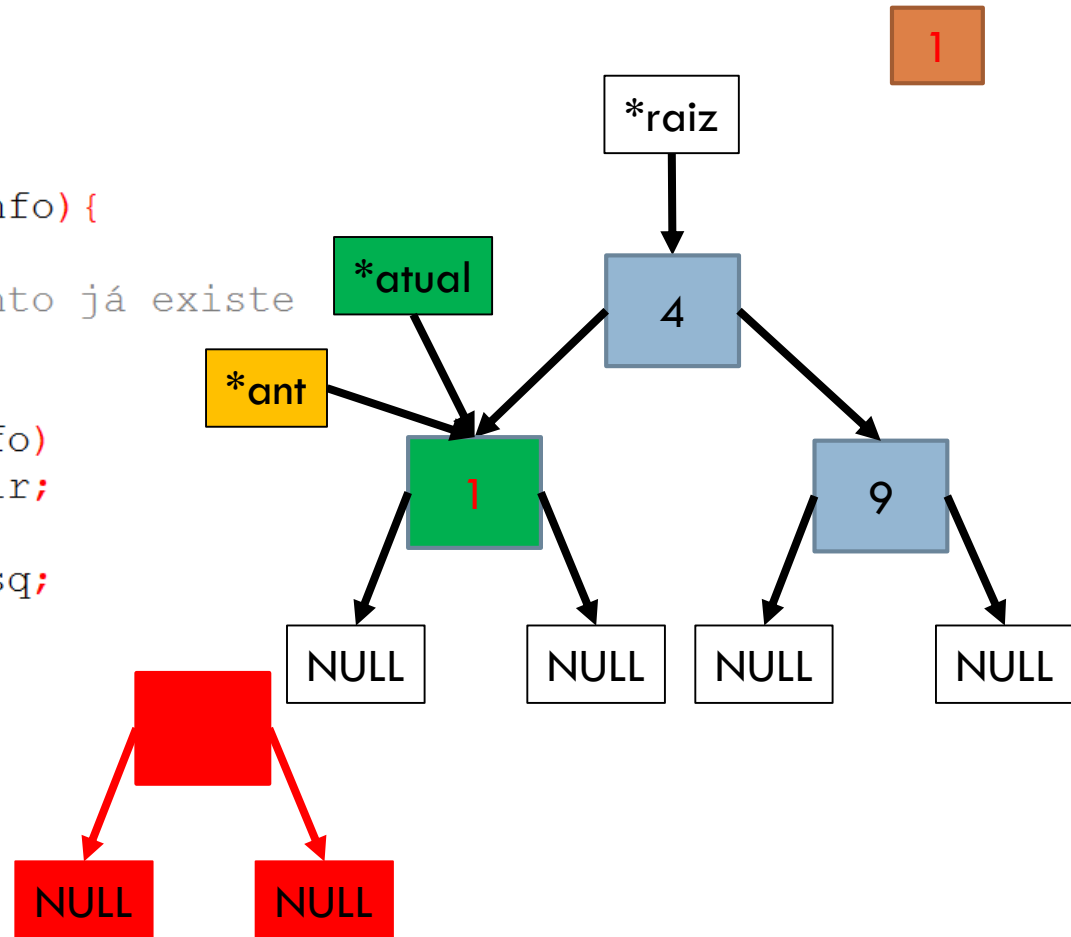
        if(valor > atual->info)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    if(valor > ant->info)
        ant->dir = novo;
    else
        ant->esq = novo;
}
return 1;
}
```



Inserção em ABB

91

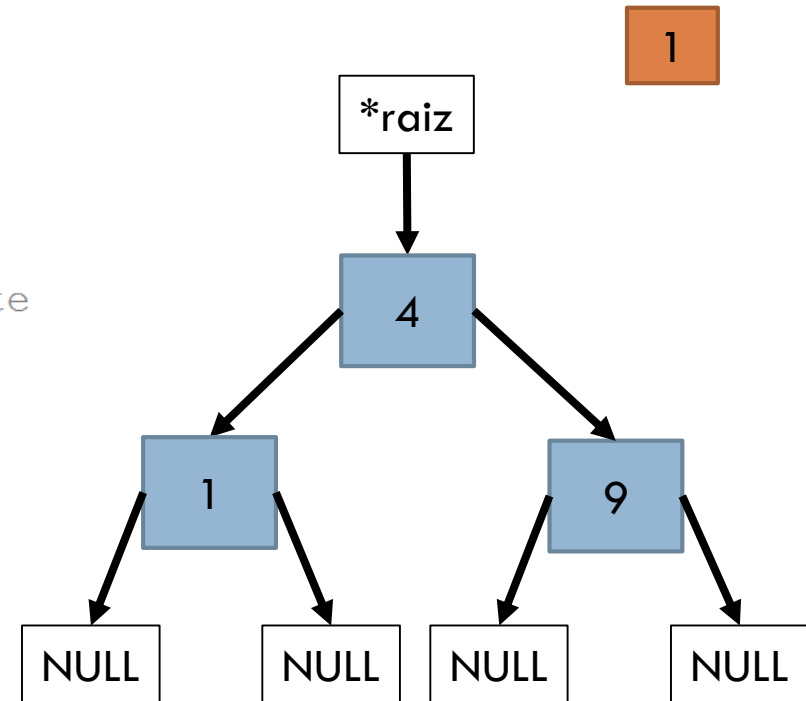
```
else{  
    [...]  
    struct NO* atual = *raiz;  
    struct NO* ant = NULL;  
    while(atual != NULL){  
        ant = atual;  
        if(valor == atual->info){  
            → free(novo);  
            return 0; //elemento já existe  
        }  
        if(valor > atual->info)  
            atual = atual->dir;  
        else  
            atual = atual->esq;  
    }  
    if(valor > ant->info)  
        ant->dir = novo;  
    else  
        ant->esq = novo;  
}  
return 1;  
}
```



Inserção em ABB

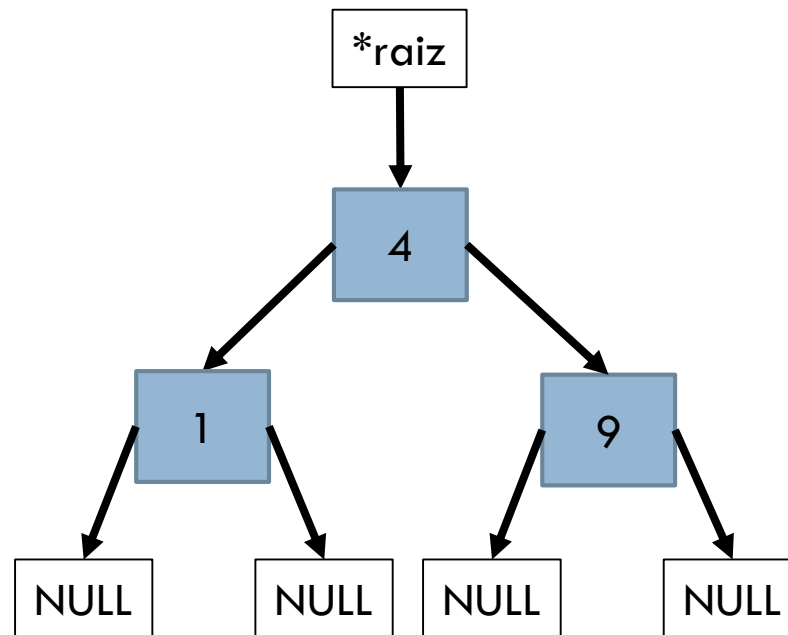
92

```
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

93



Busca em ABB

94

- 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

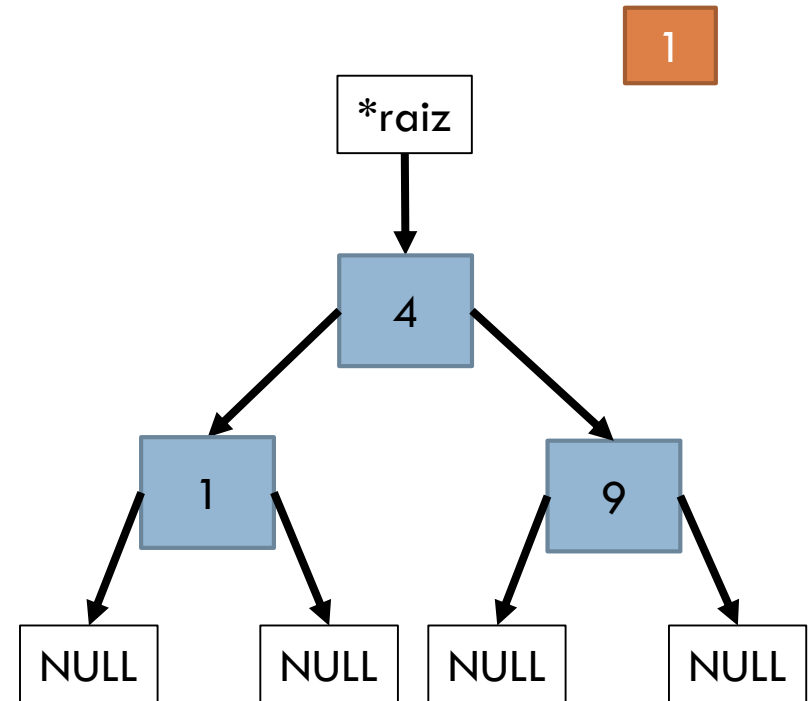
95

- 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

96

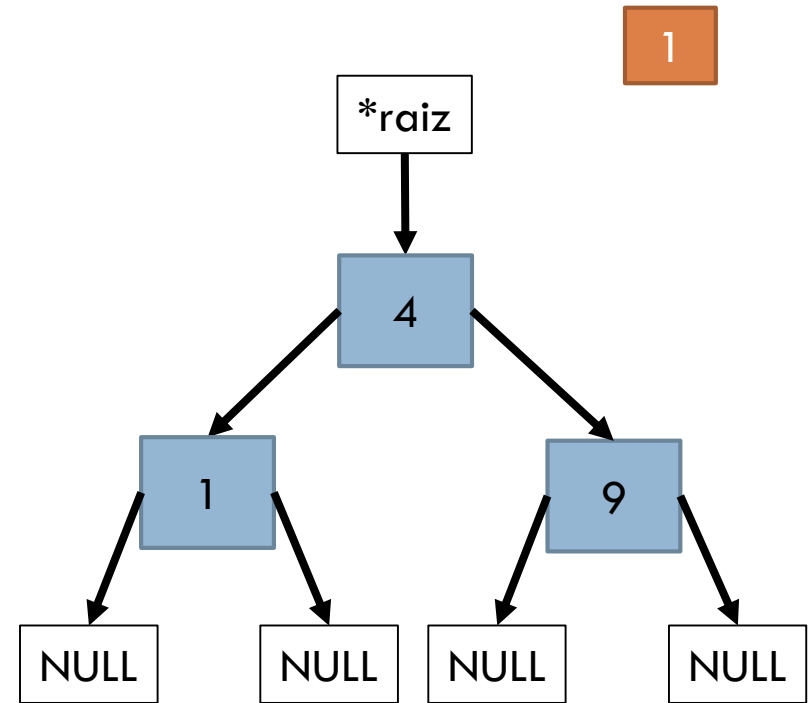
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

97

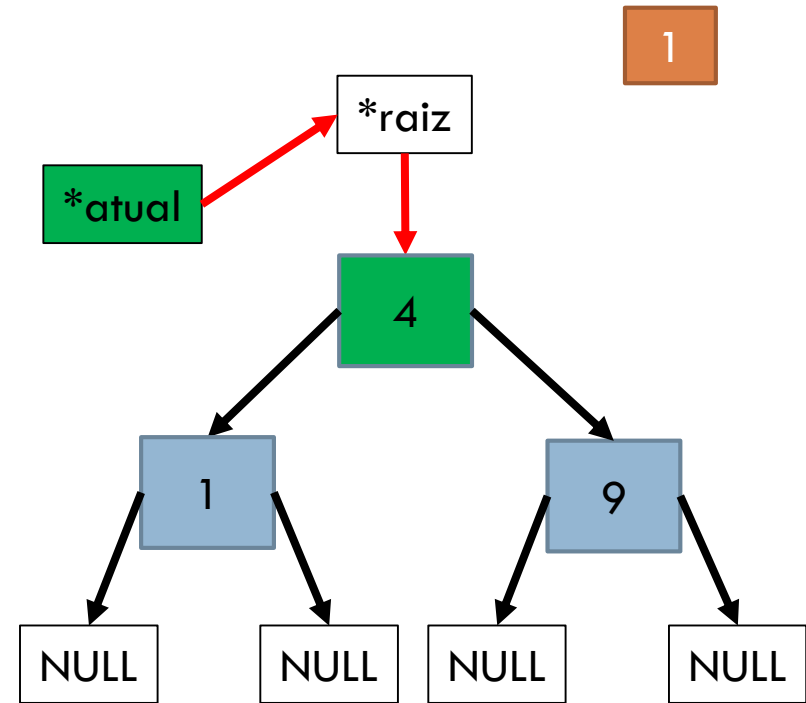
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){  
    → if(raiz == NULL)  
        return 0;  
    struct NO* atual = *raiz;  
    while(atual != NULL){  
        if(valor == atual->info){  
            return atual;  
        }  
        if(valor > atual->info)  
            atual = atual->dir;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



Busca em ABB

98

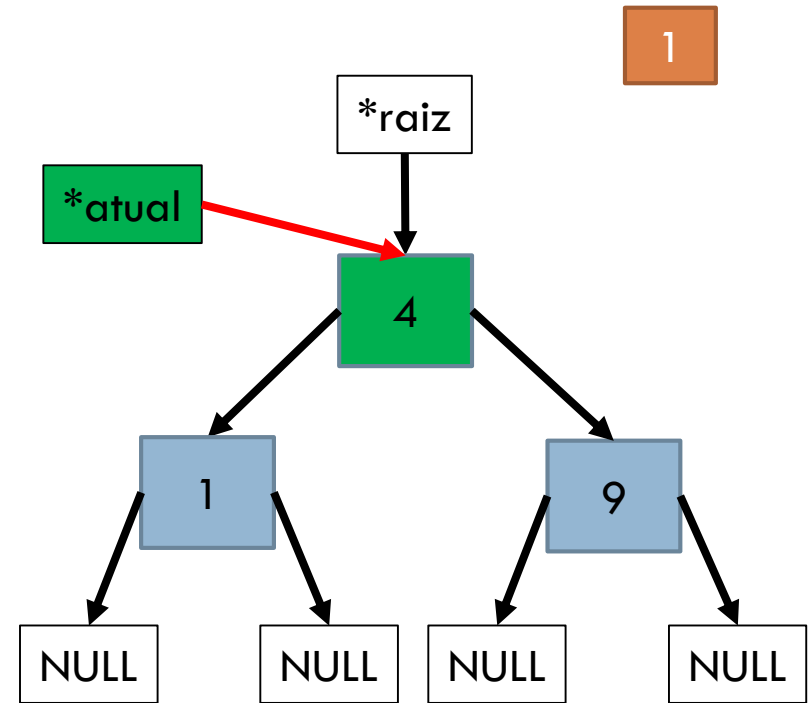
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    → struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

99

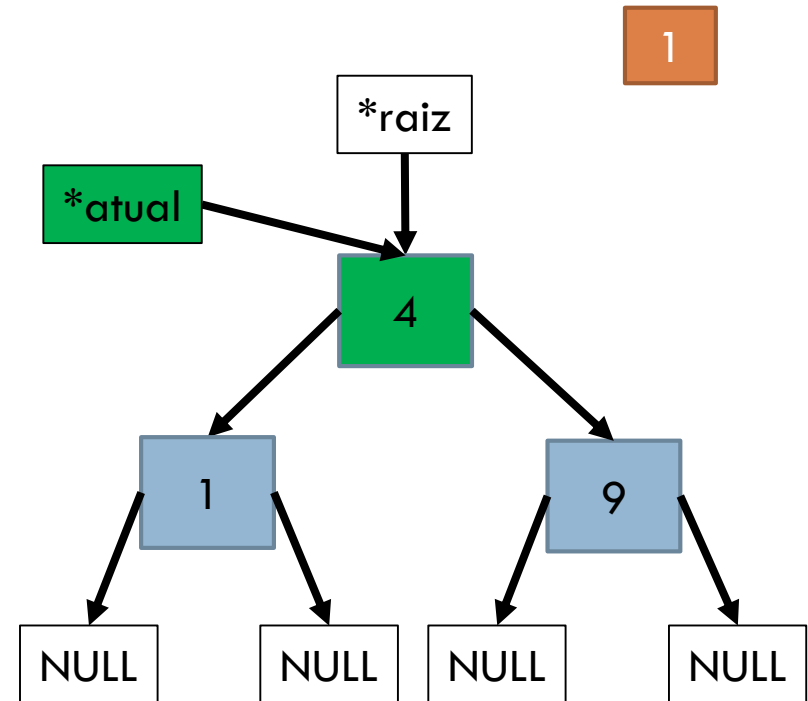
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    → struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

100

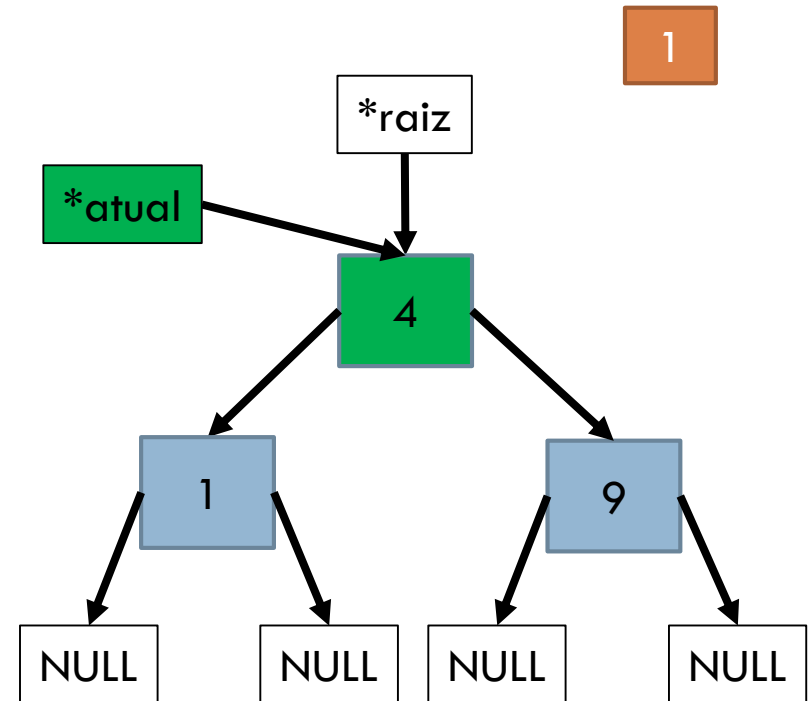
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    → while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

101

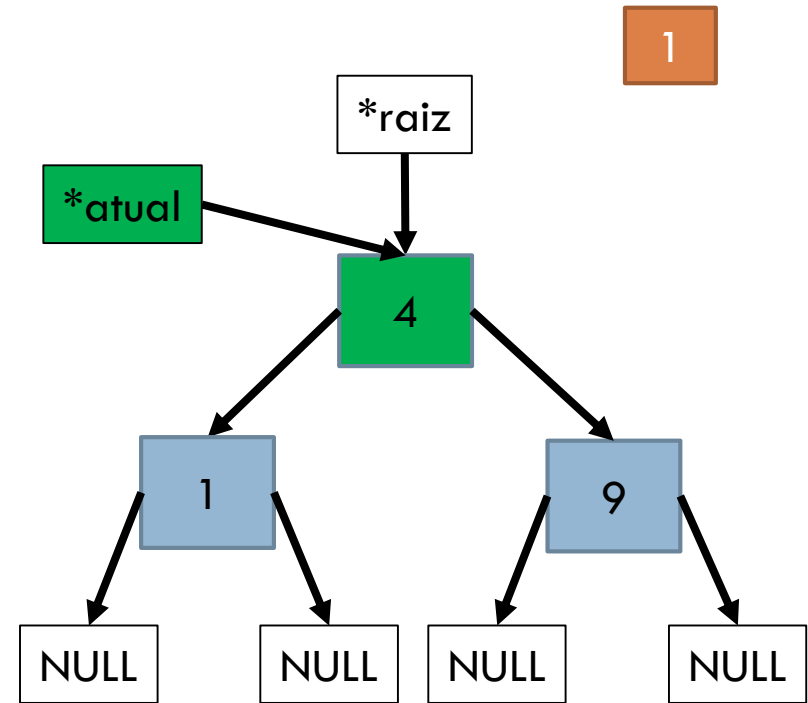
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        → if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

102

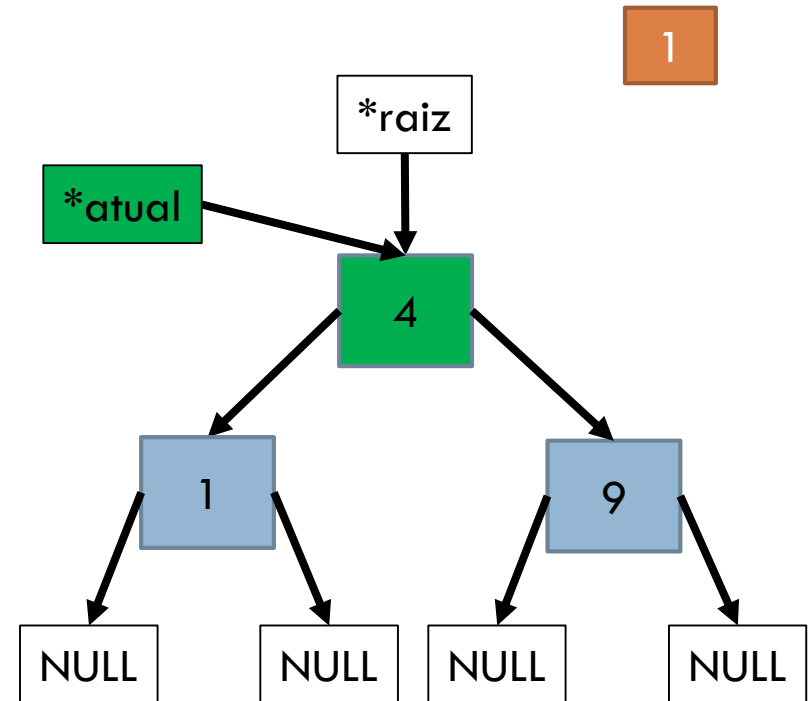
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* atual = *raiz;  
    while(atual != NULL){  
        if(valor == atual->info){  
            return atual;  
        }  
        → if(valor > atual->info)  
            atual = atual->dir;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



Busca em ABB

103

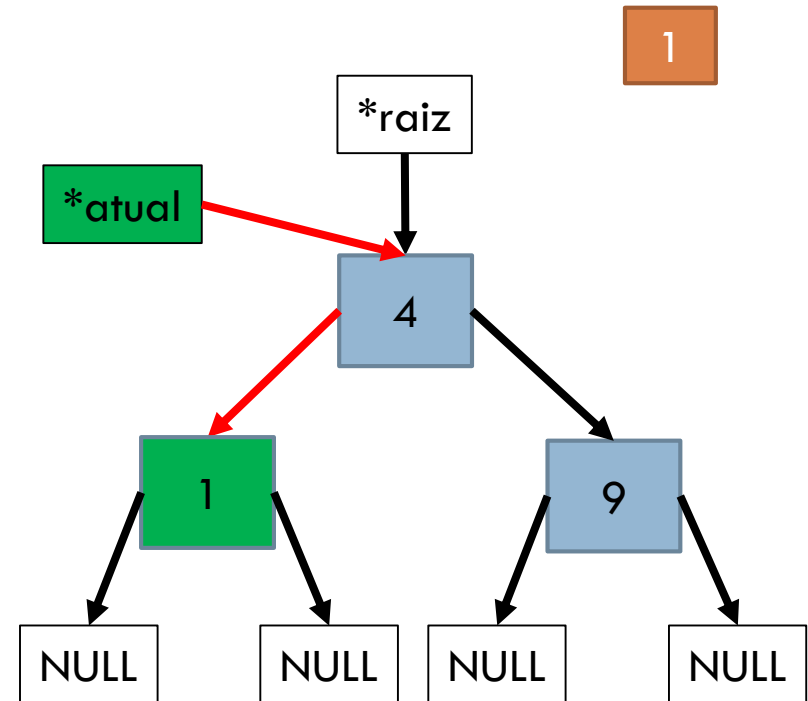
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        → else
  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



Busca em ABB

104

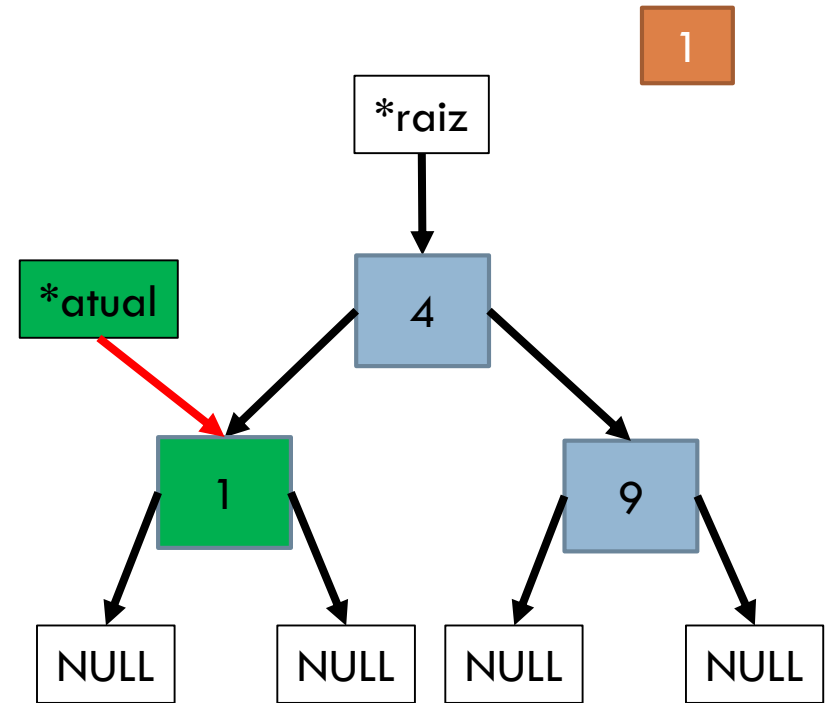
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

105

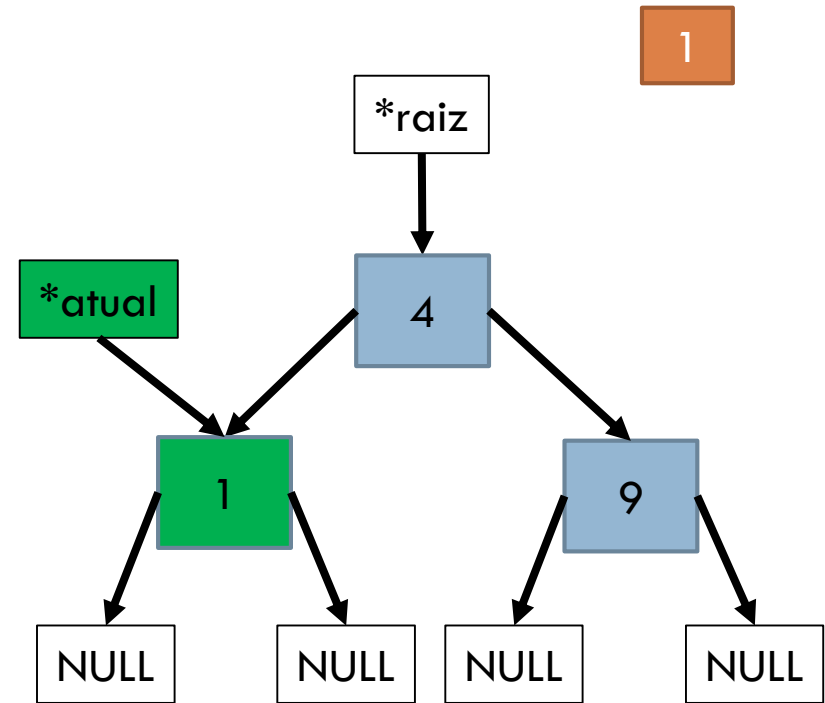
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

106

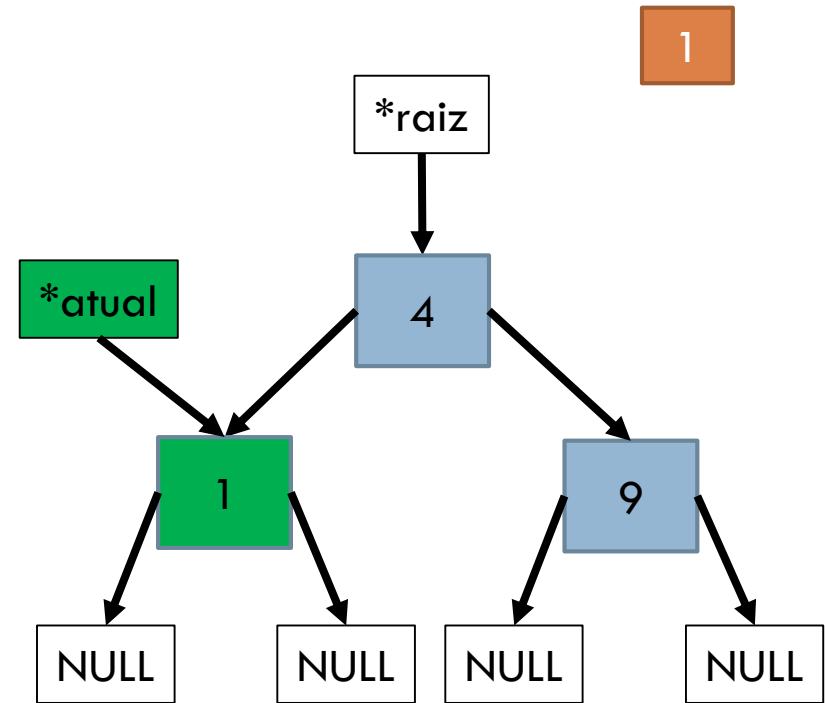
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    → while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

107

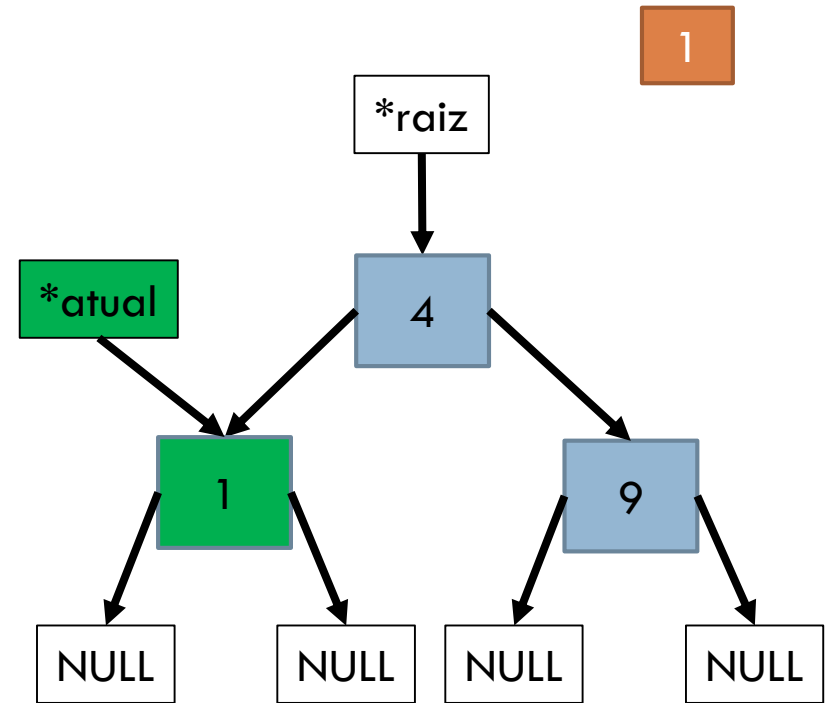
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        → if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

108

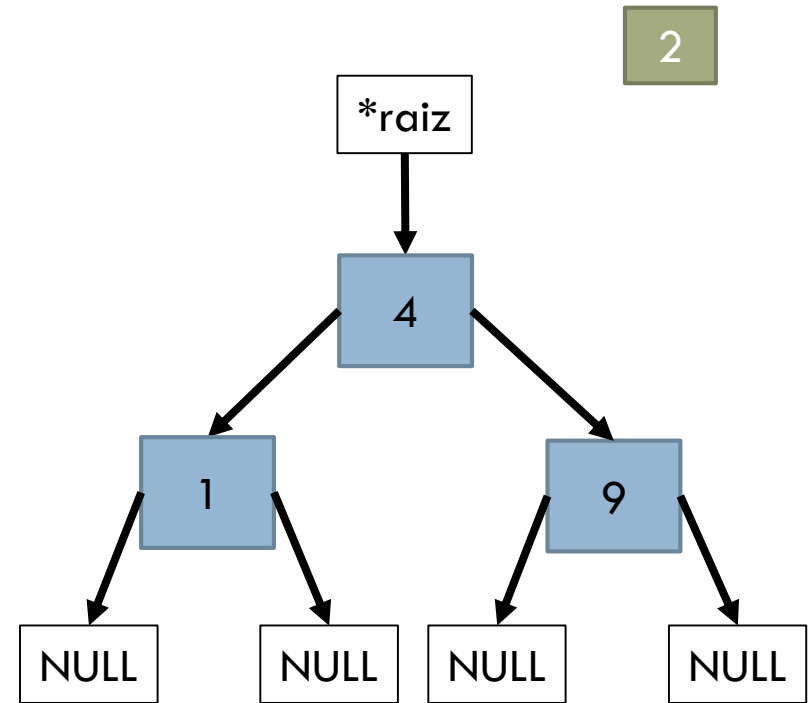
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            → return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

109

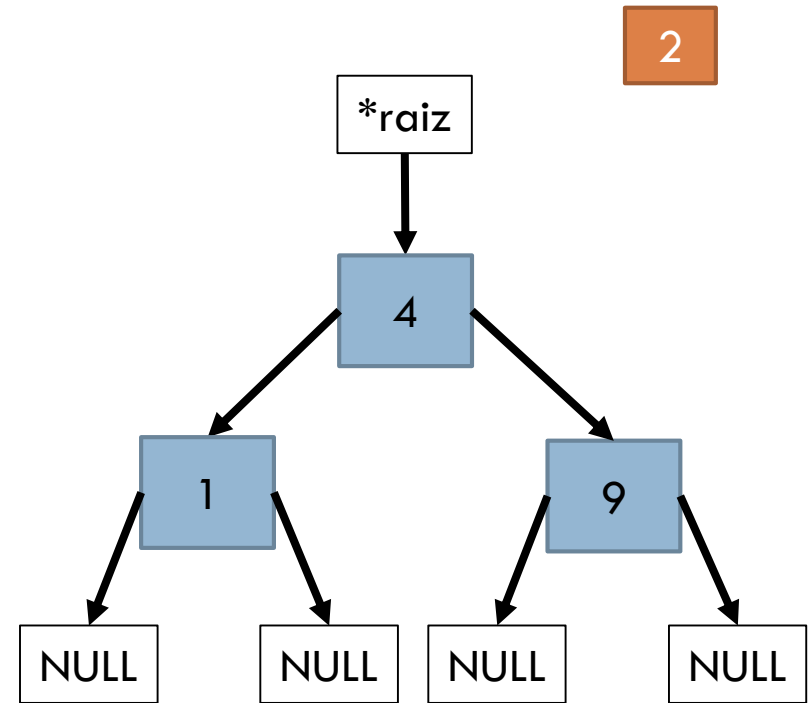
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

110

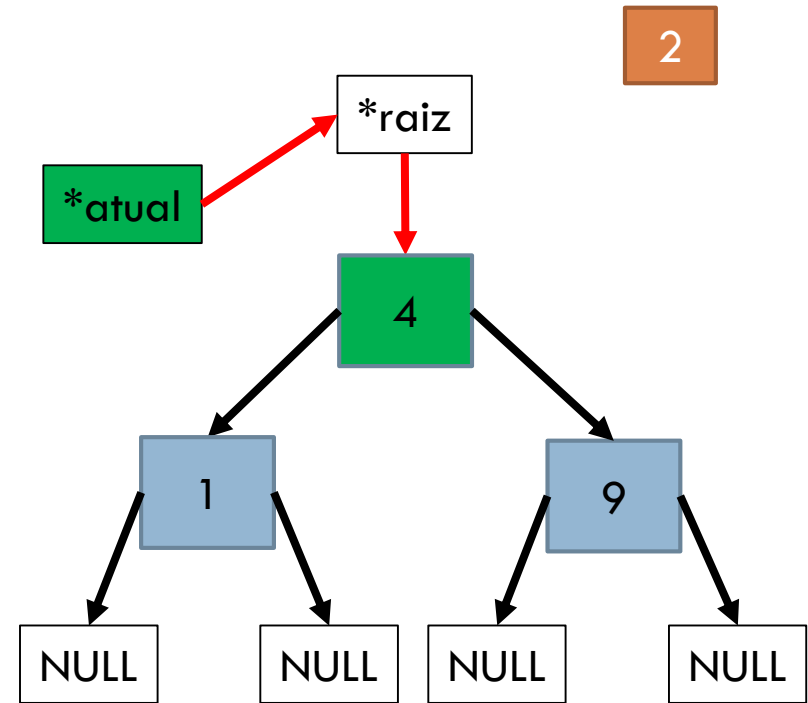
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){  
    → if(raiz == NULL)  
        return 0;  
    struct NO* atual = *raiz;  
    while(atual != NULL){  
        if(valor == atual->info){  
            return atual;  
        }  
        if(valor > atual->info)  
            atual = atual->dir;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



Busca em ABB

111

```
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;
}
```

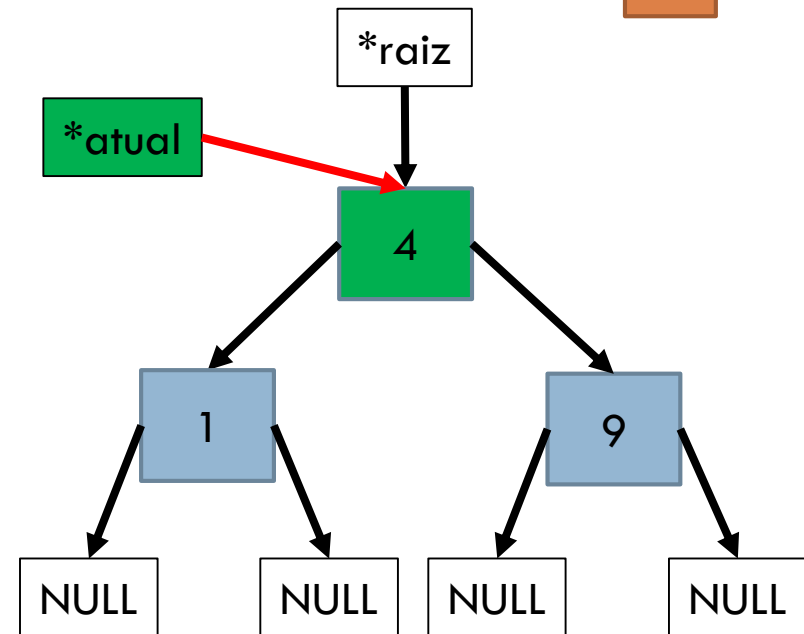


Busca em ABB

112

2

```
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;  
}
```

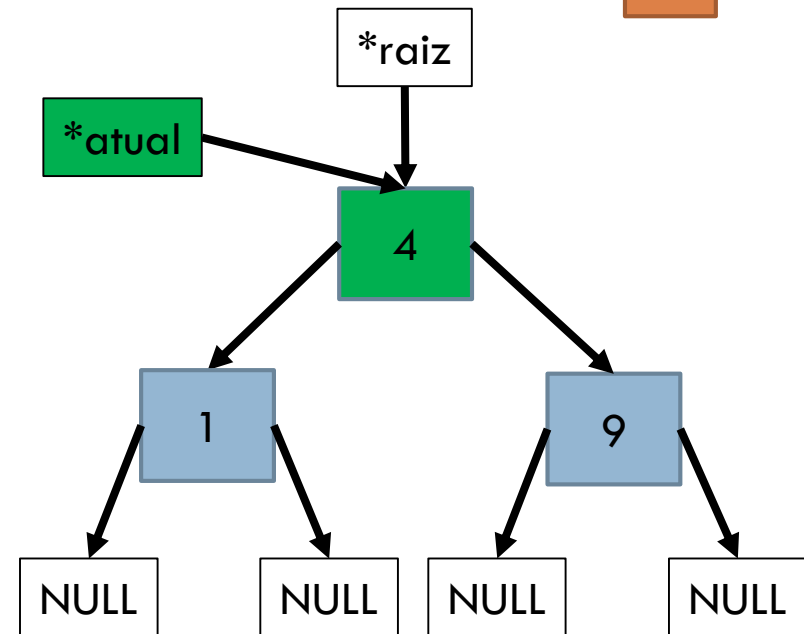


Busca em ABB

113

2

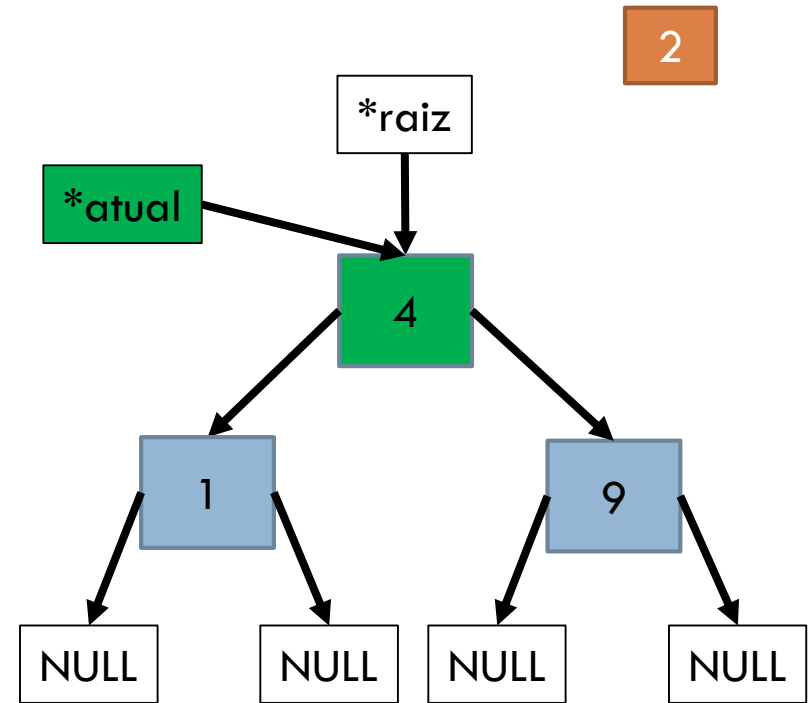
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* atual = *raiz;  
    → while(atual != NULL){  
        if(valor == atual->info){  
            return atual;  
        }  
        if(valor > atual->info)  
            atual = atual->dir;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



Busca em ABB

114

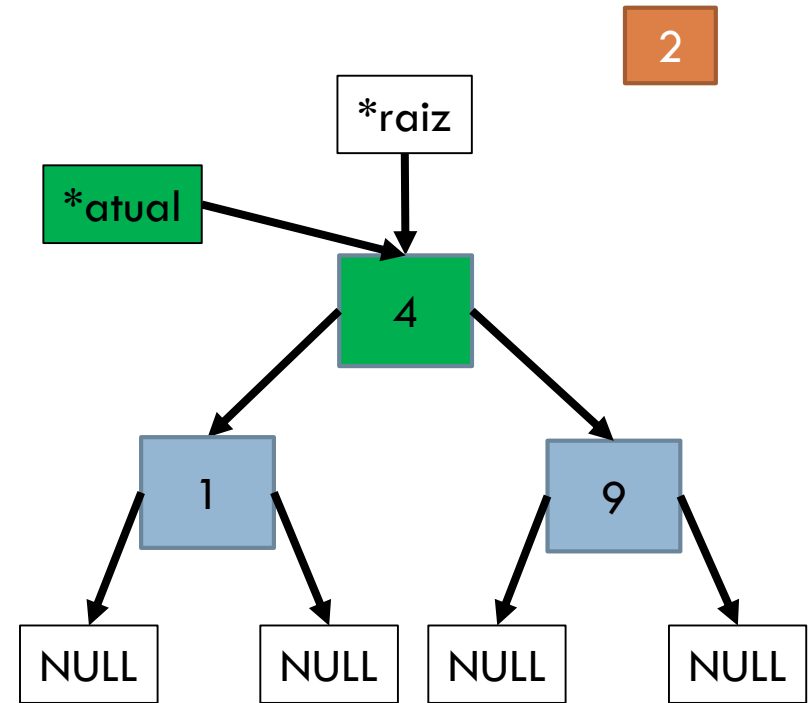
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        → if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

115

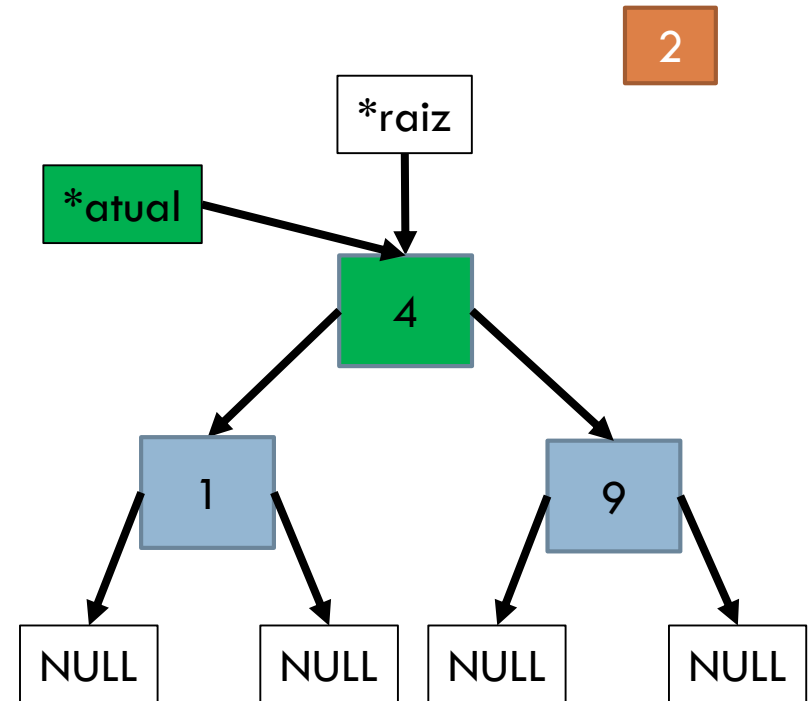
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        → if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

116

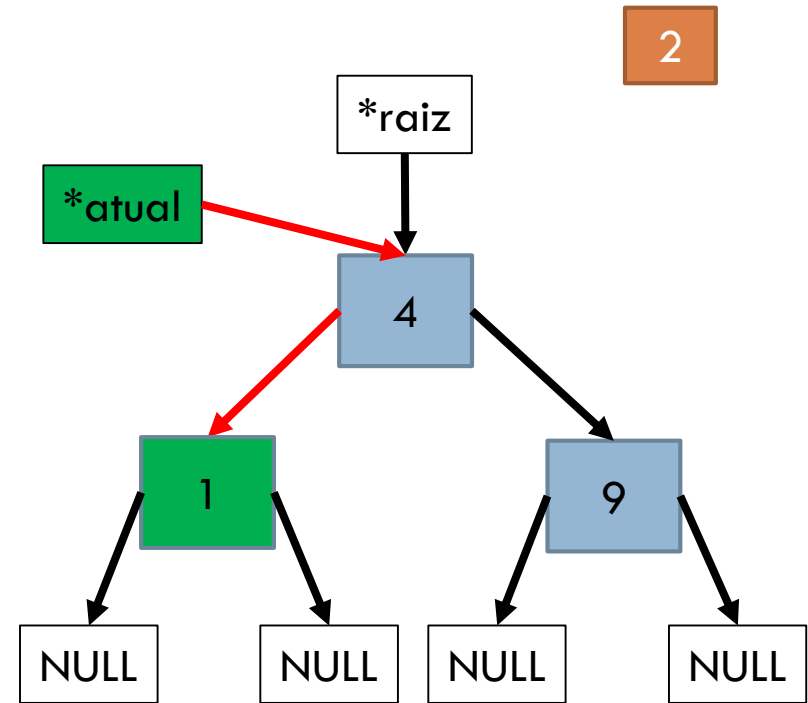
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        → else
  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



Busca em ABB

117

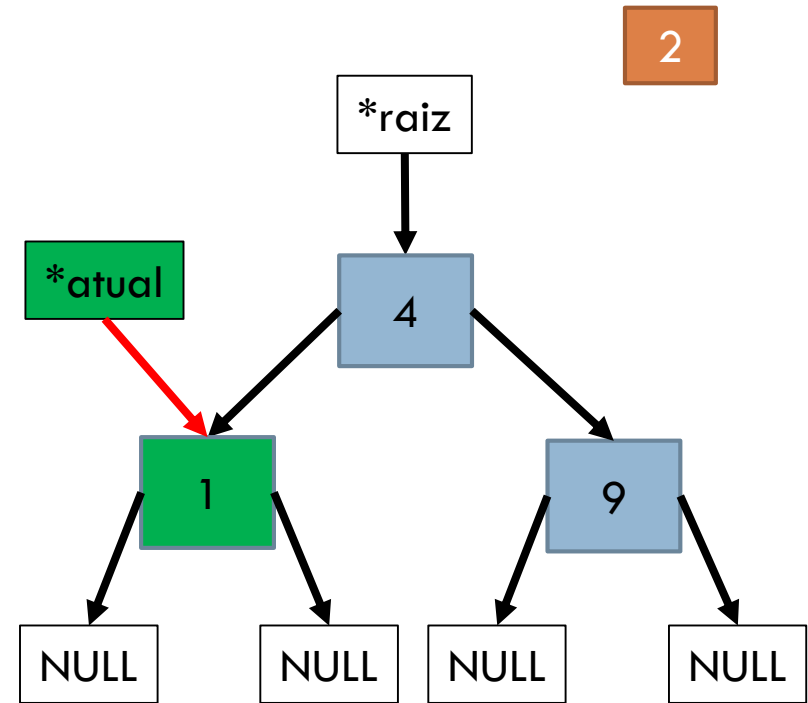
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

118

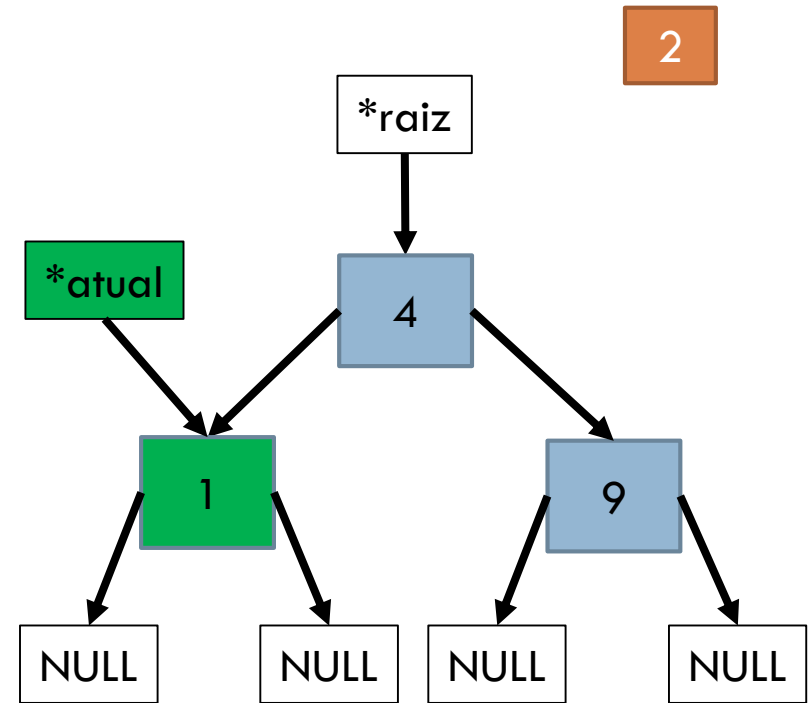
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
    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



Busca em ABB

119

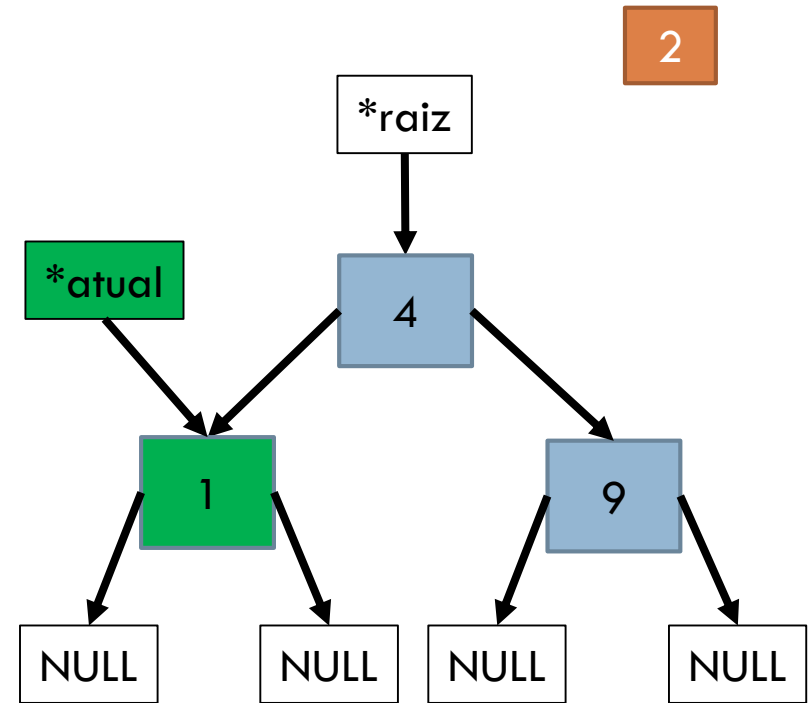
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    → while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

120

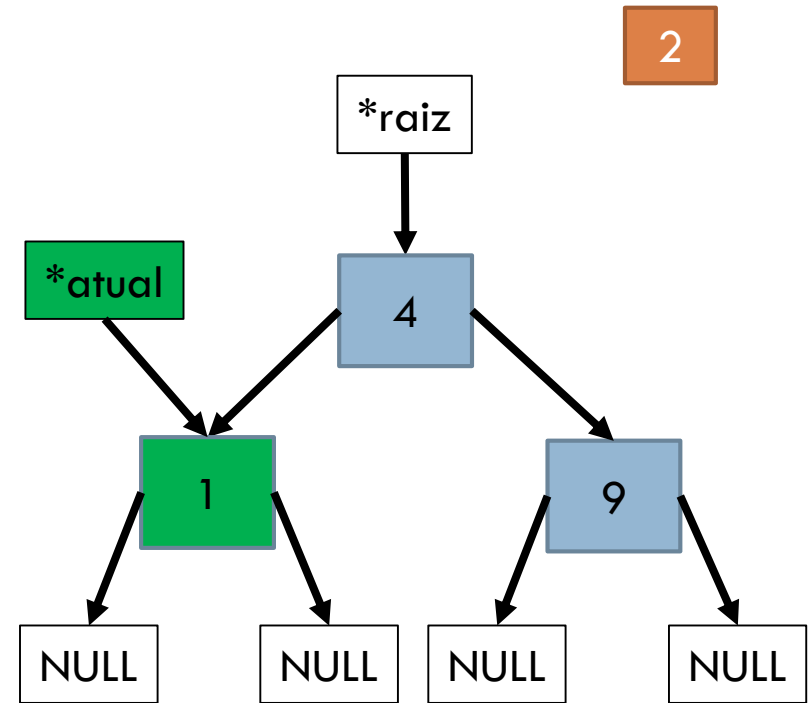
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        → if(valor == atual->info){
            return atual;
        }
        if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

121

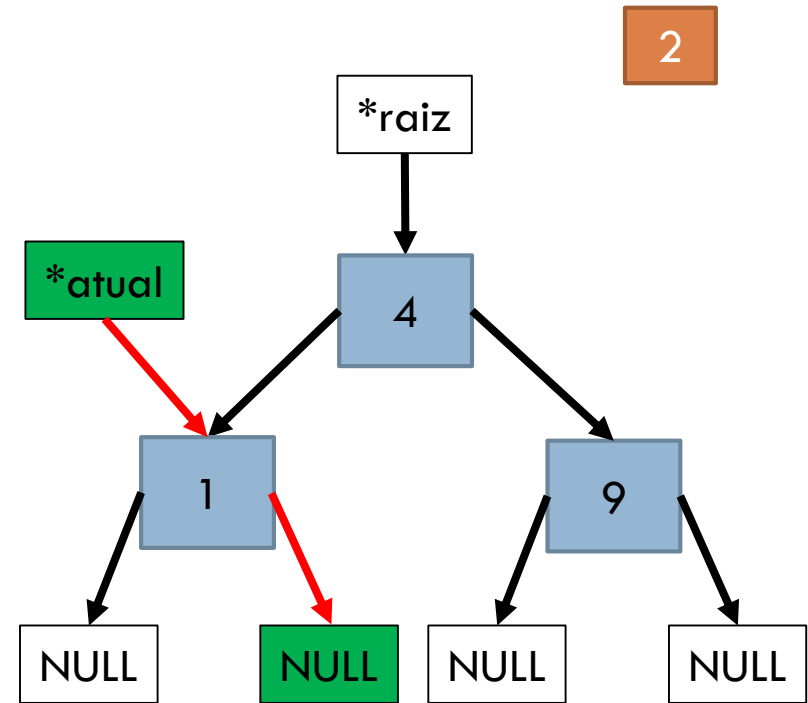
```
struct NO* busca_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            return atual;
        }
        → if(valor > atual->info)
            atual = atual->dir;
        else
  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



Busca em ABB

122

```
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;
}
```

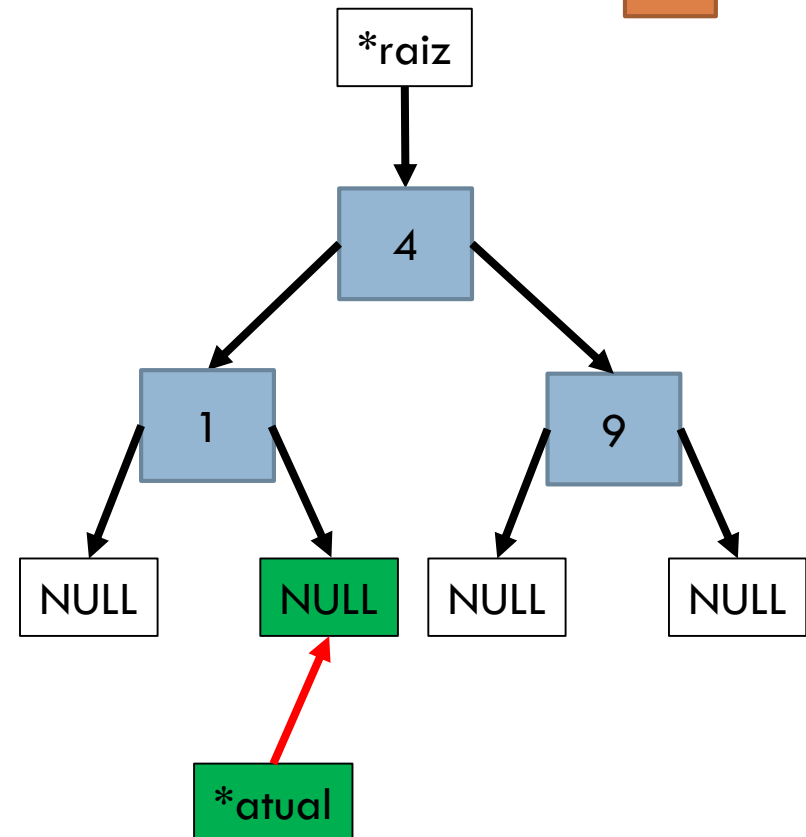


Busca em ABB

123

2

```
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;
}
```

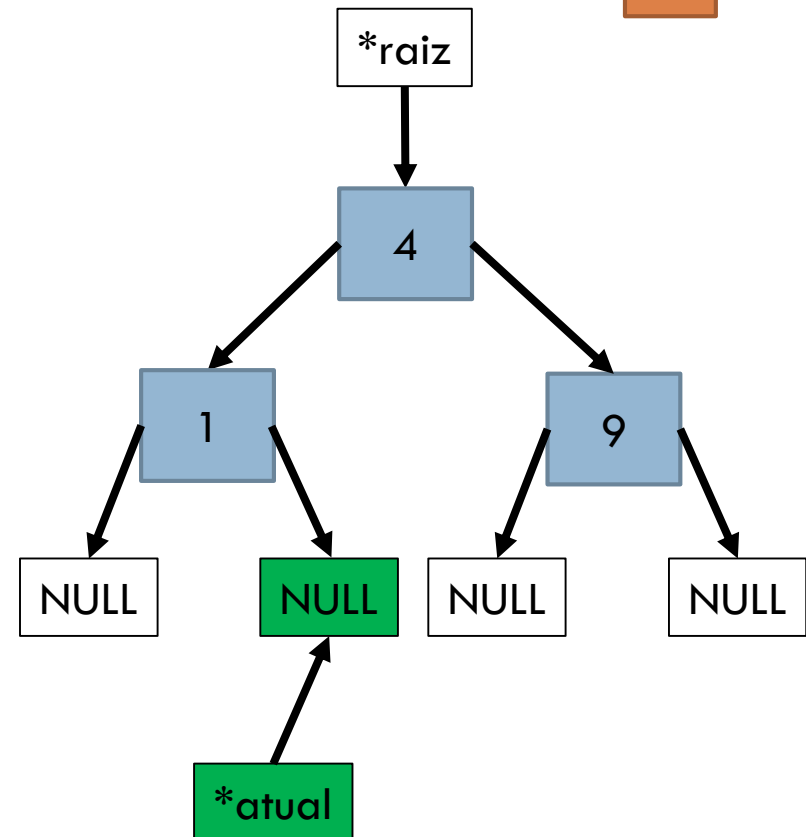


Busca em ABB

124

2

```
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;  
}
```

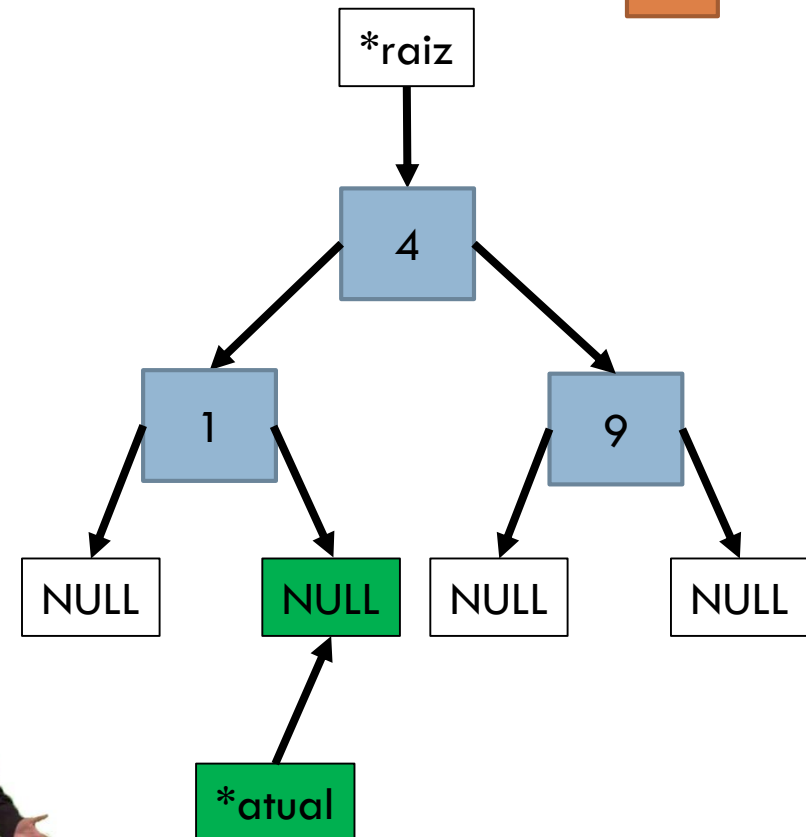


Busca em ABB

125

2

```
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

126

- 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

127

- 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

128

- 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

129

- 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

130

- 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

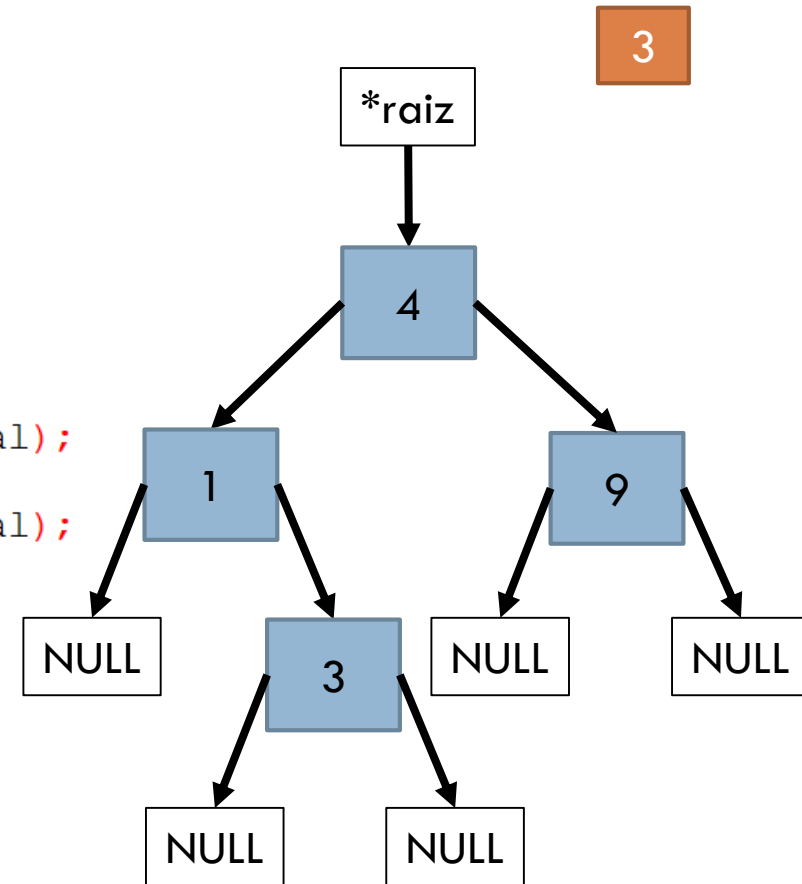
131

- 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

132

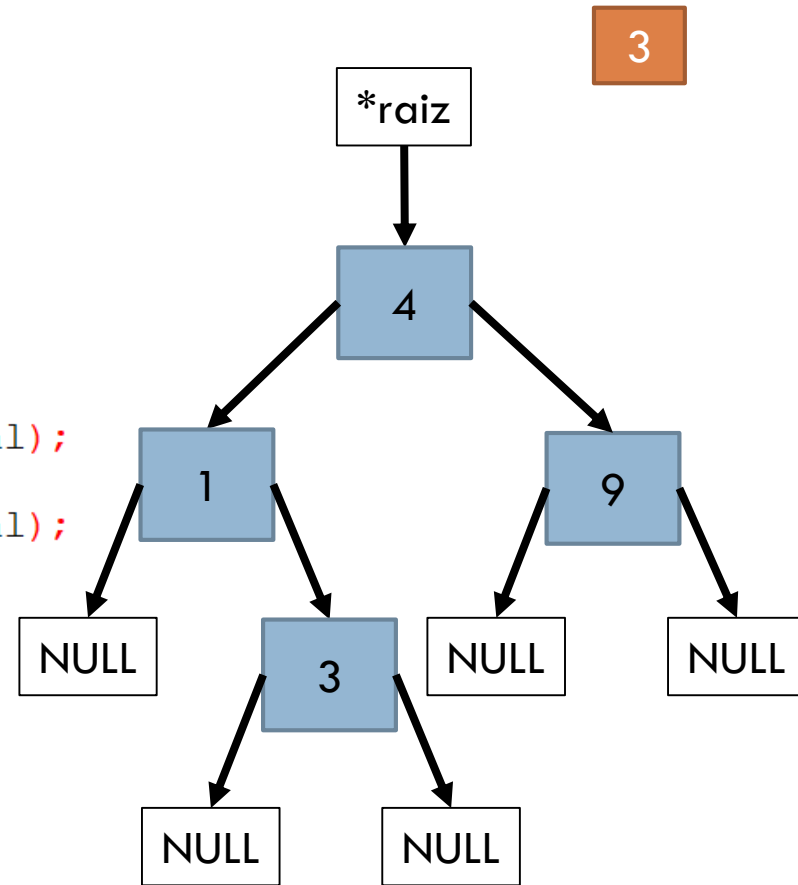
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

133

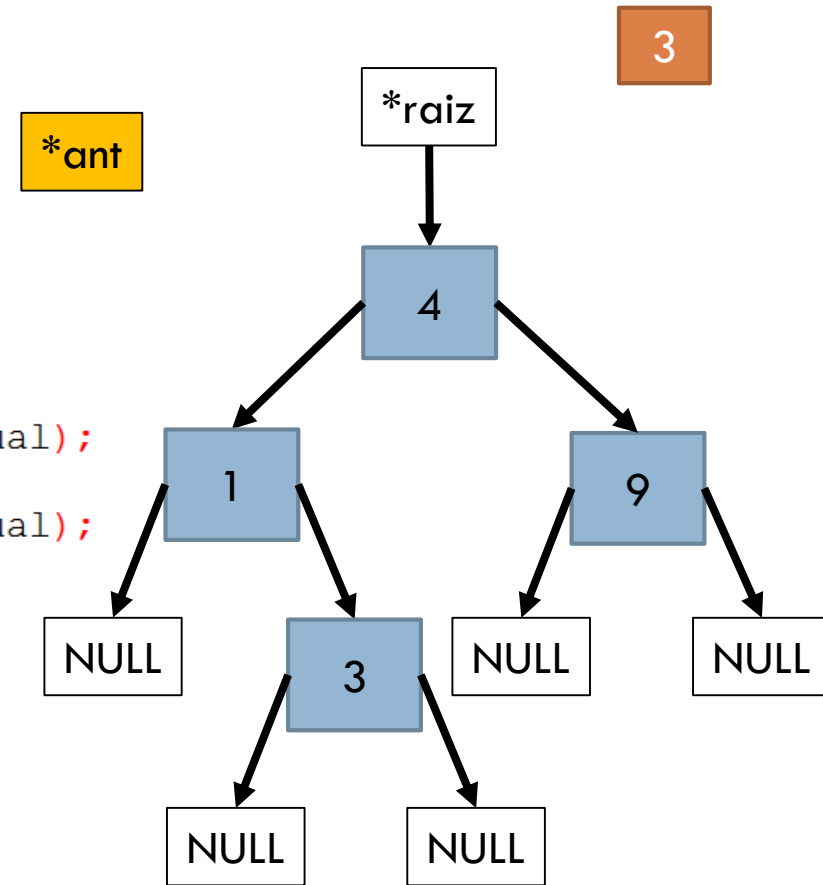
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    → if(raiz == NULL)  
        return 0;  
    struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    while(atual != NULL) {  
        if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

134

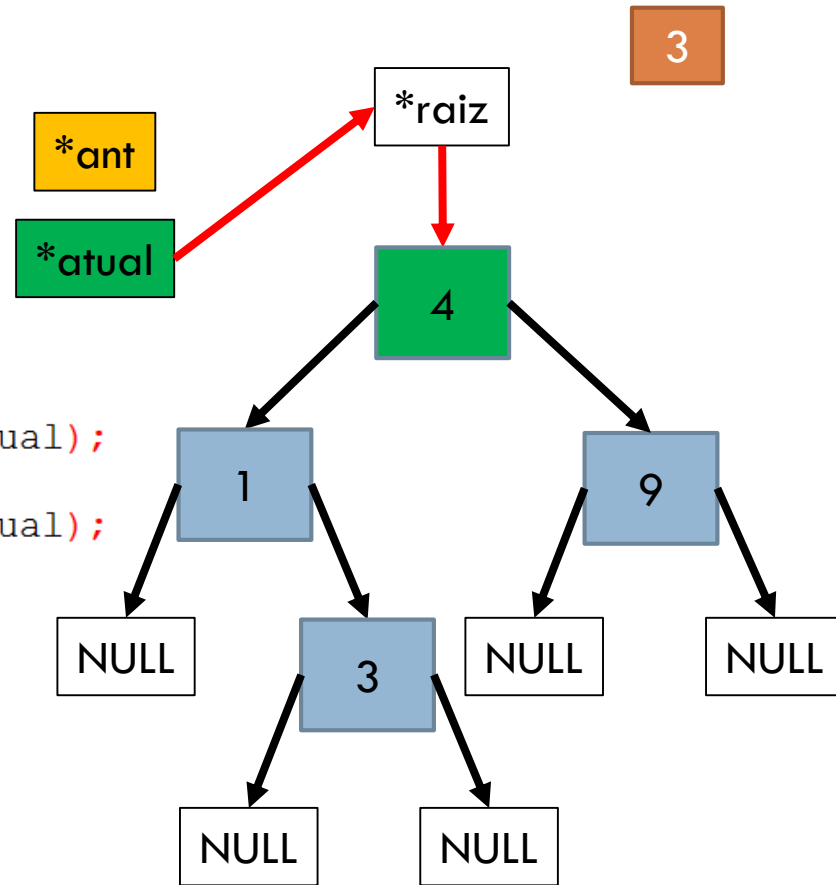
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    → struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

135

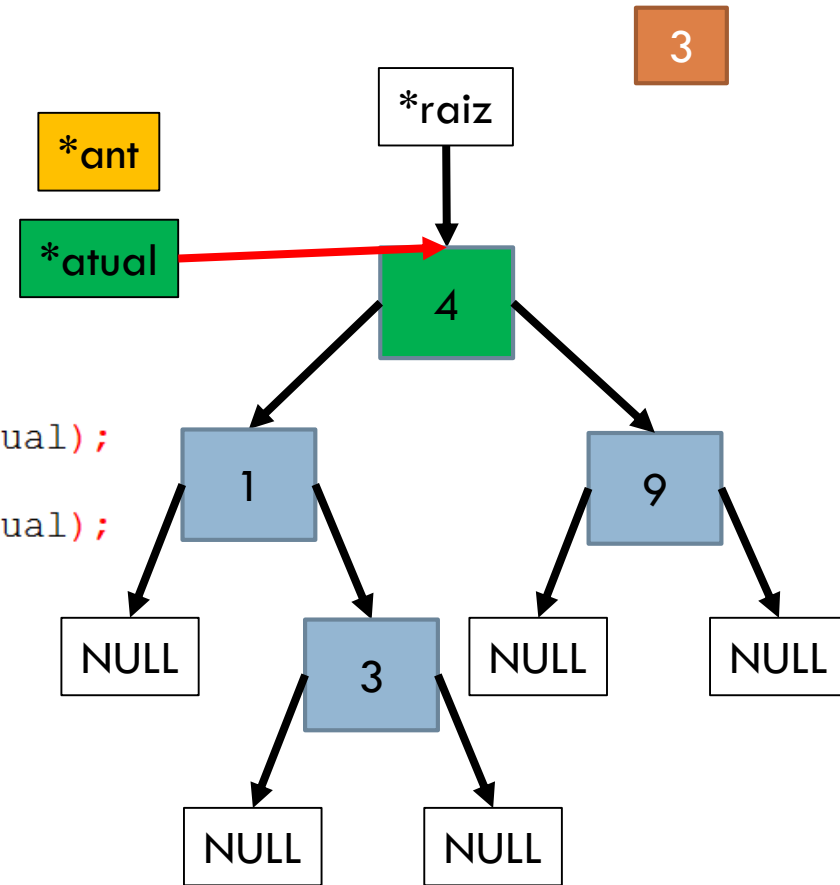
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

136

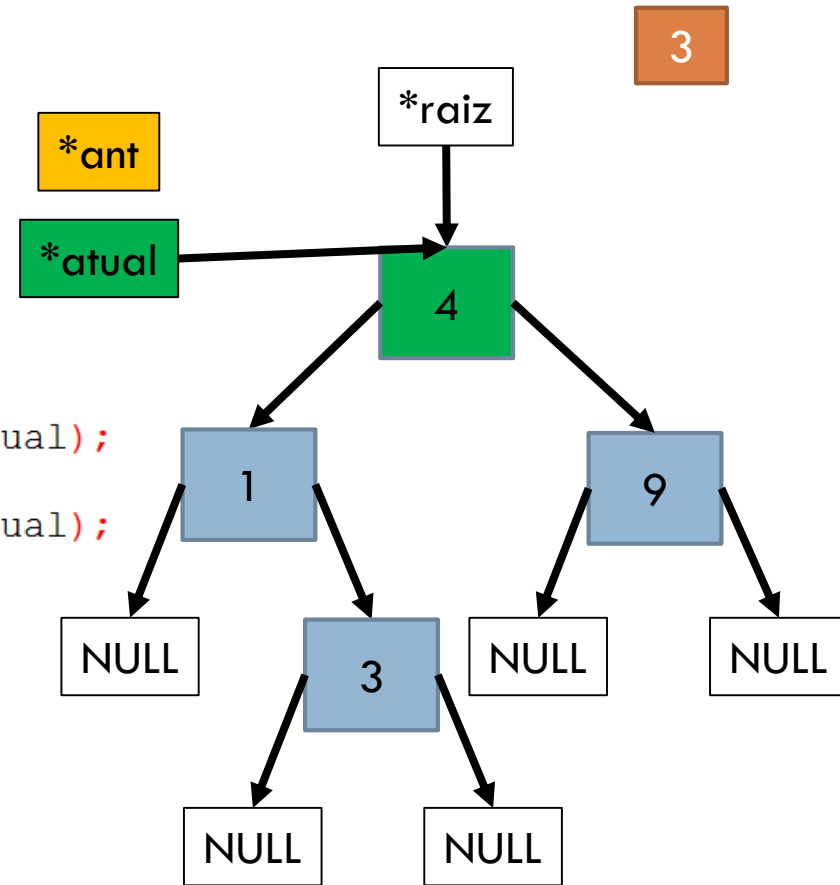
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

137

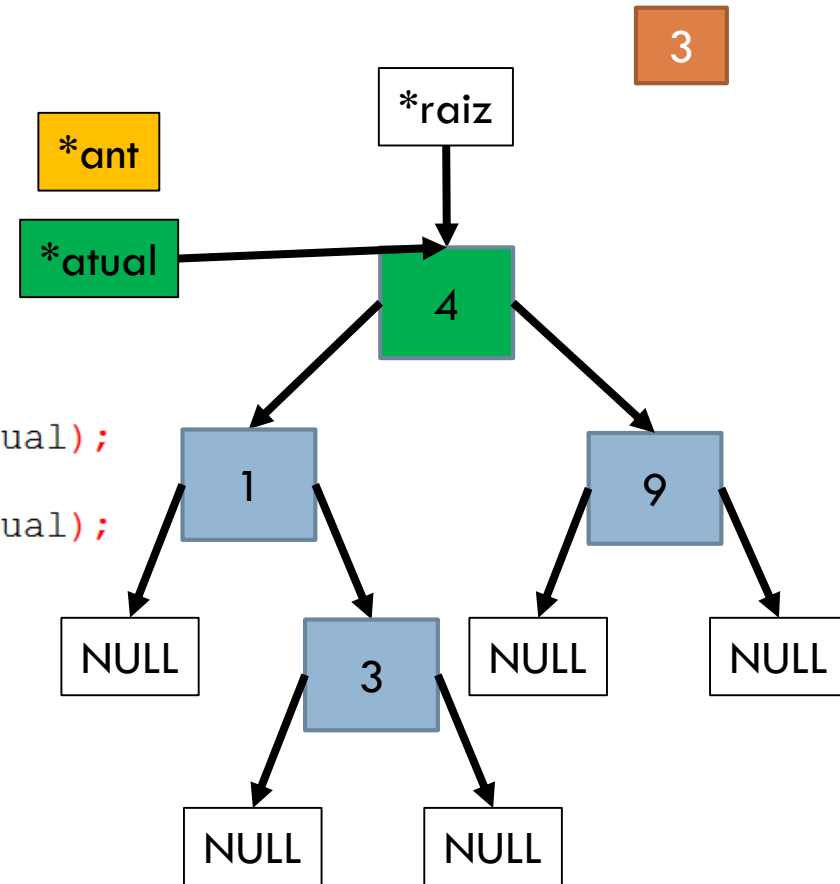
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    → while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

138

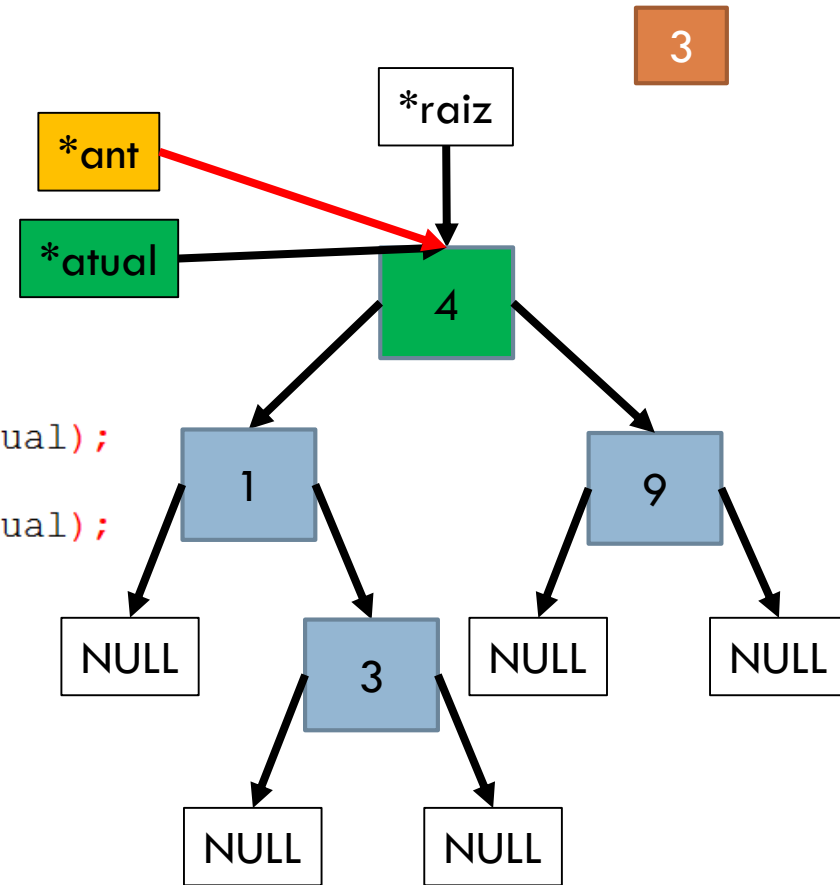
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        → if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

139

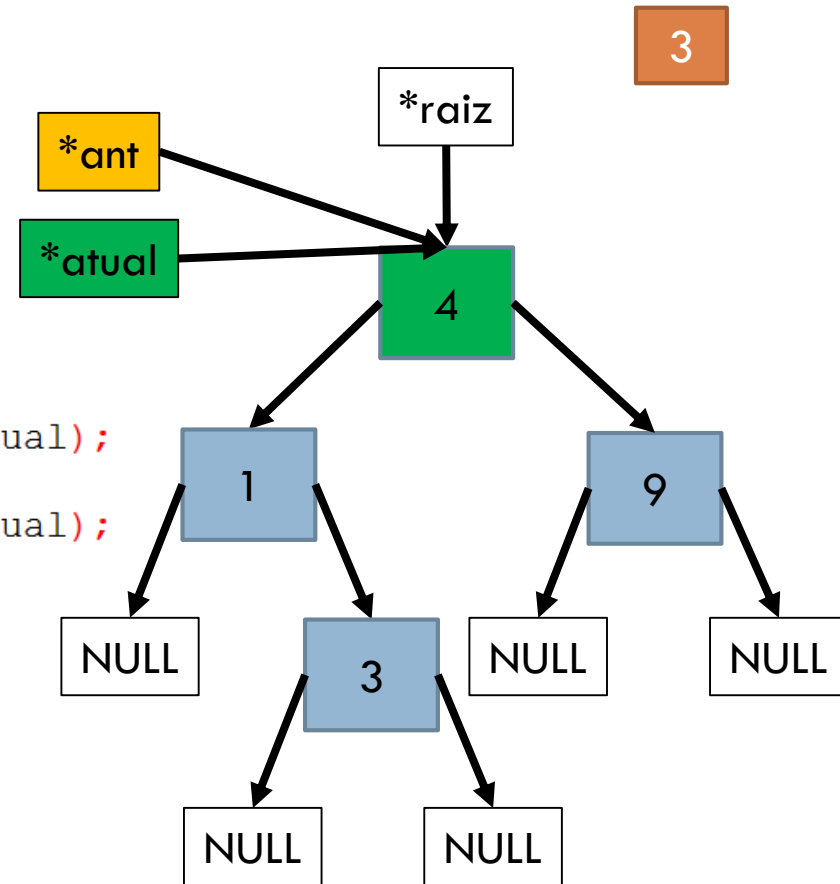
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

140

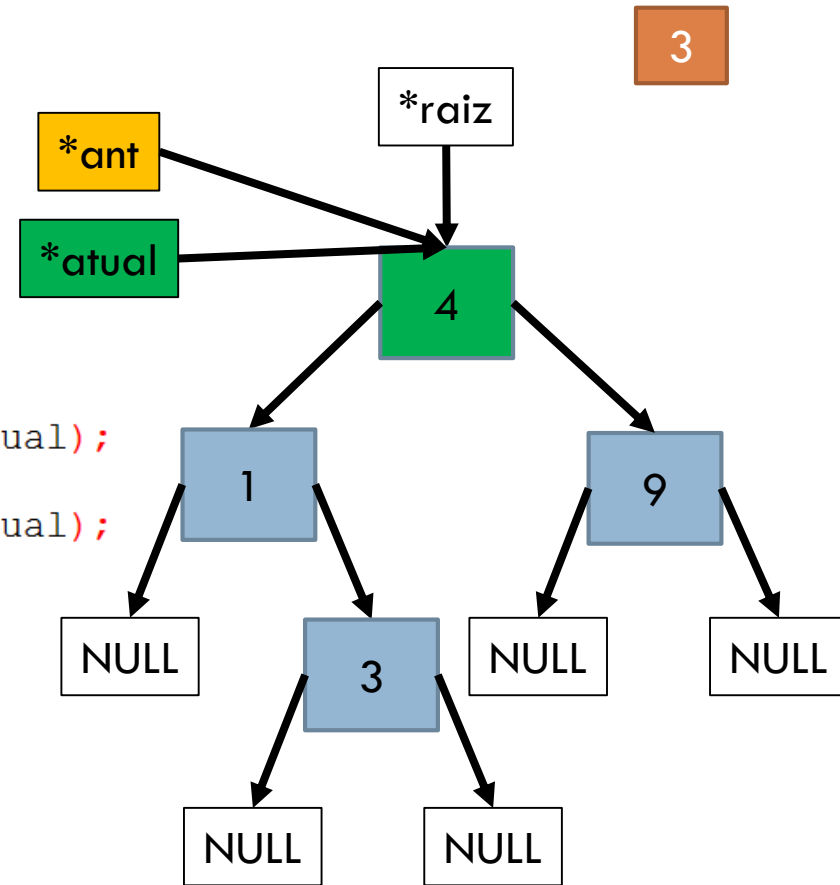
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

141

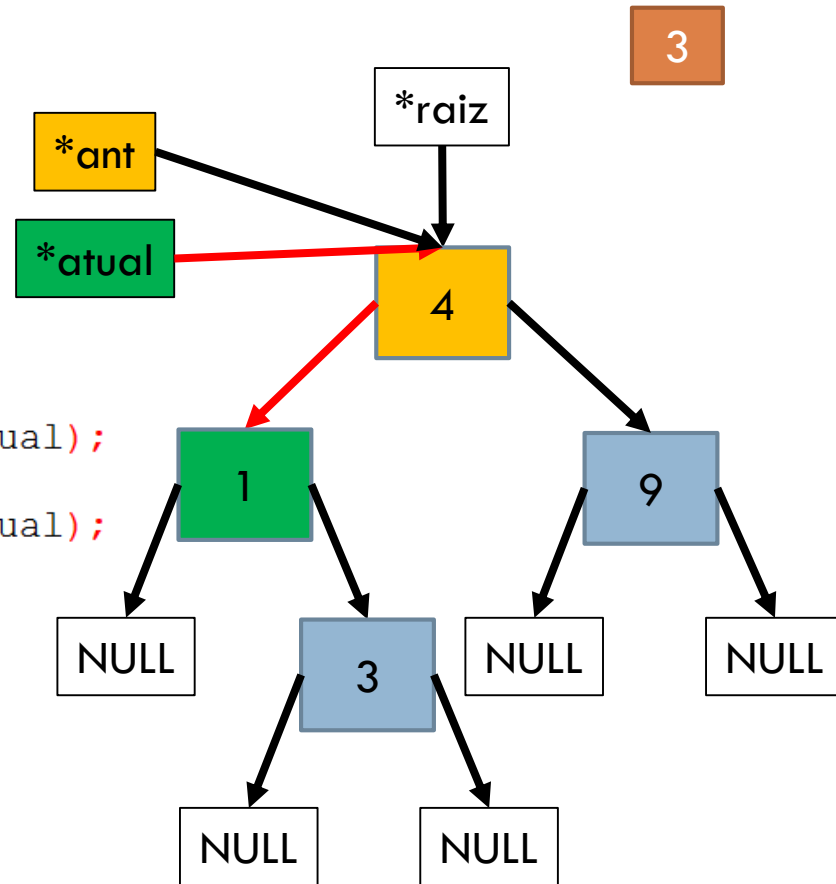
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

142

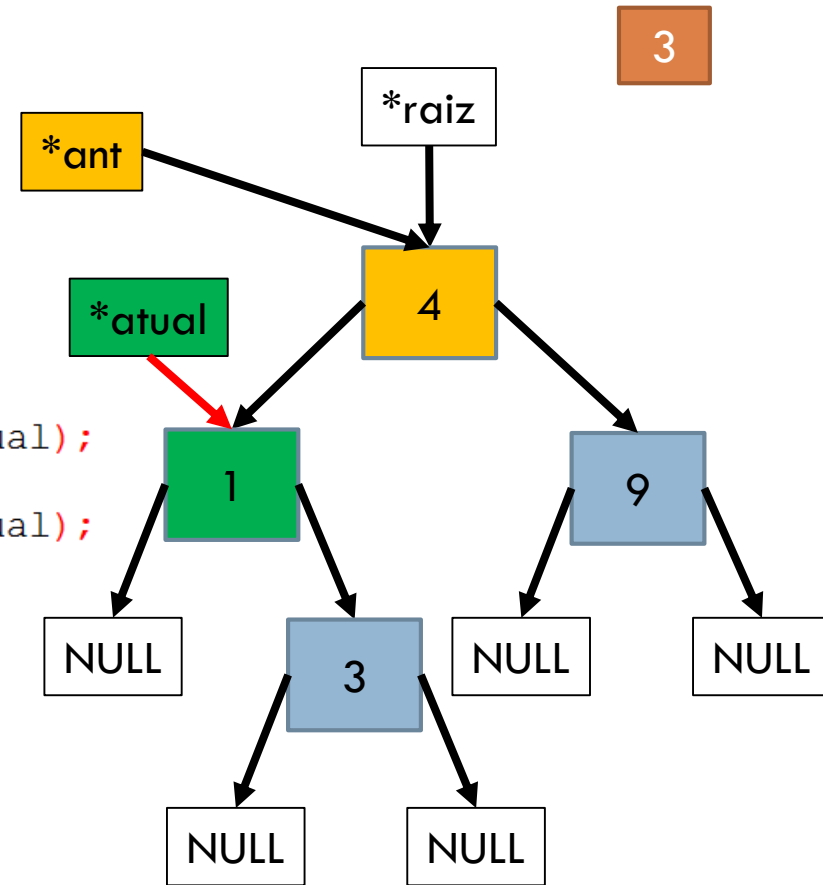
```
int remove_ArvBin(ArvBin *raiz, int valor){
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL){
        if(valor == atual->info){
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else{
  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

143

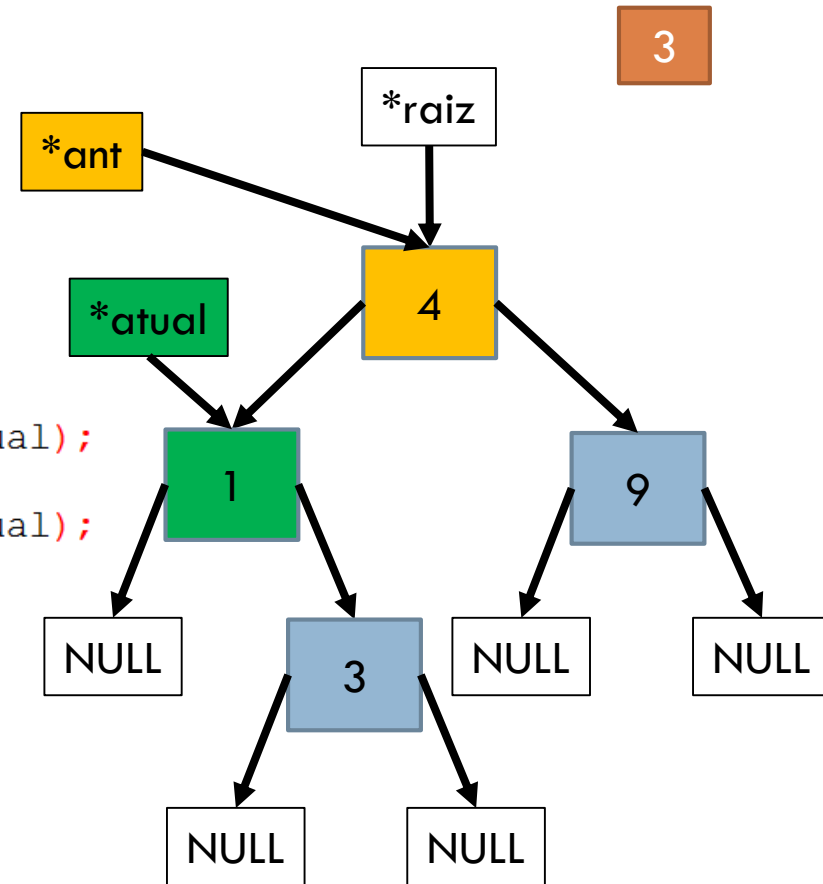
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

144

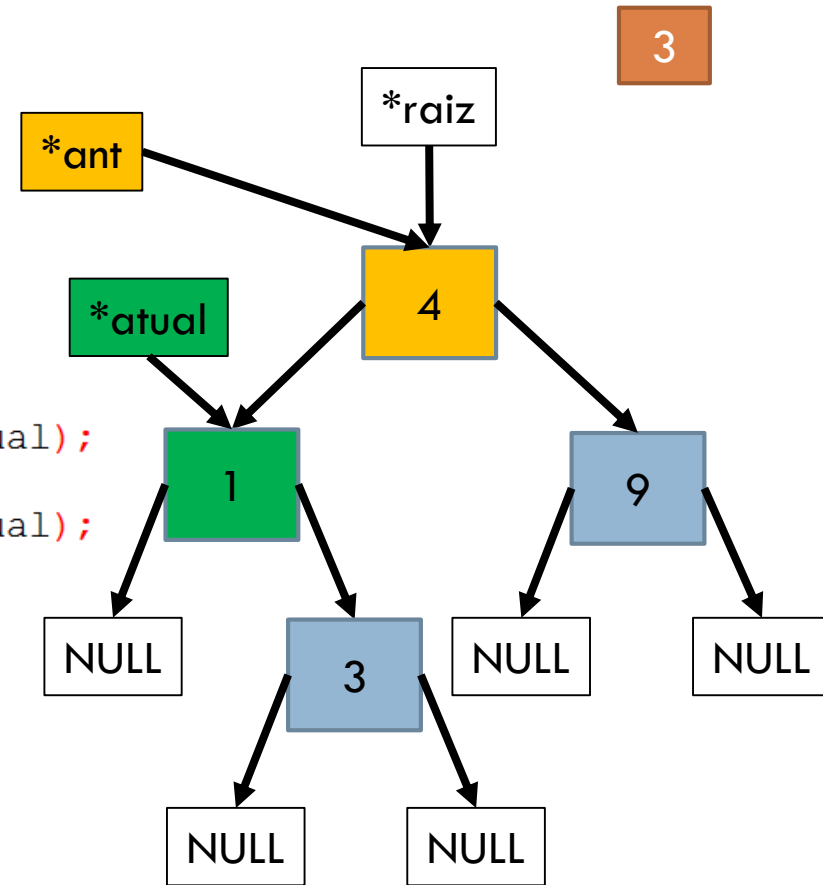
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    → while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

145

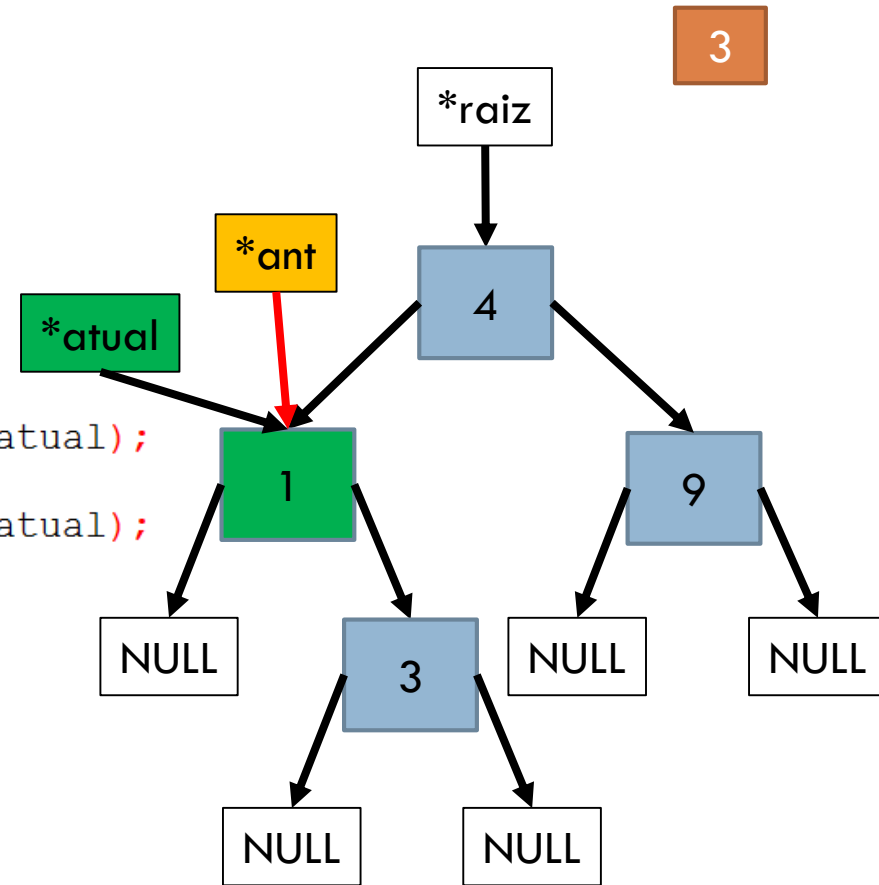
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        → if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

146

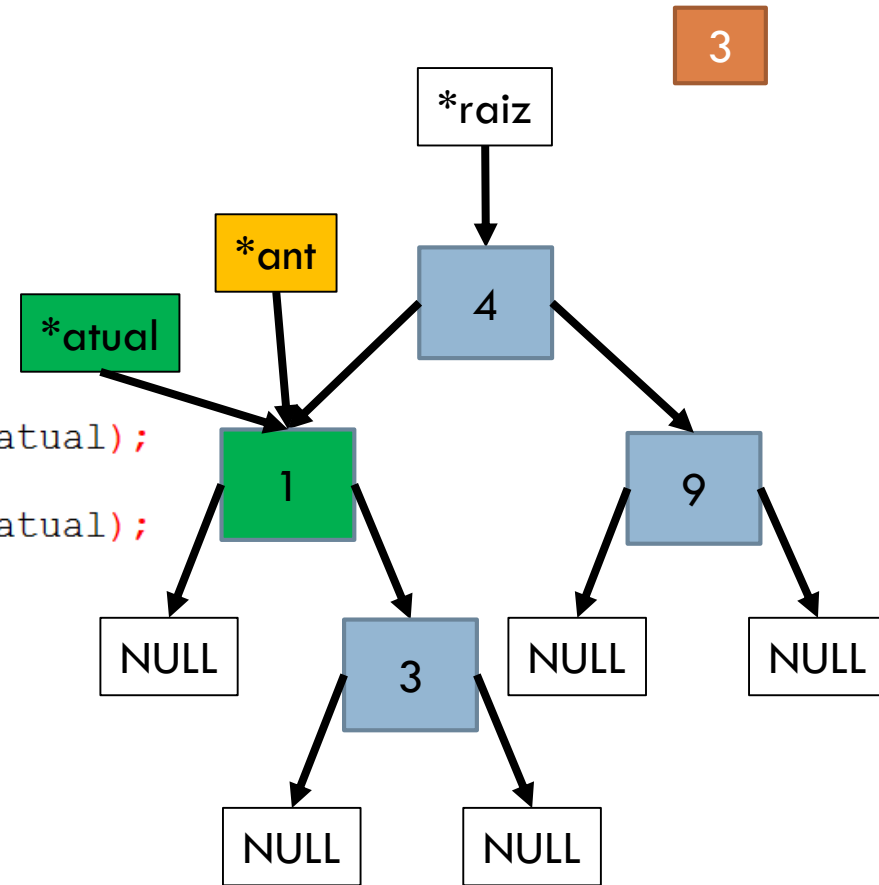
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

147

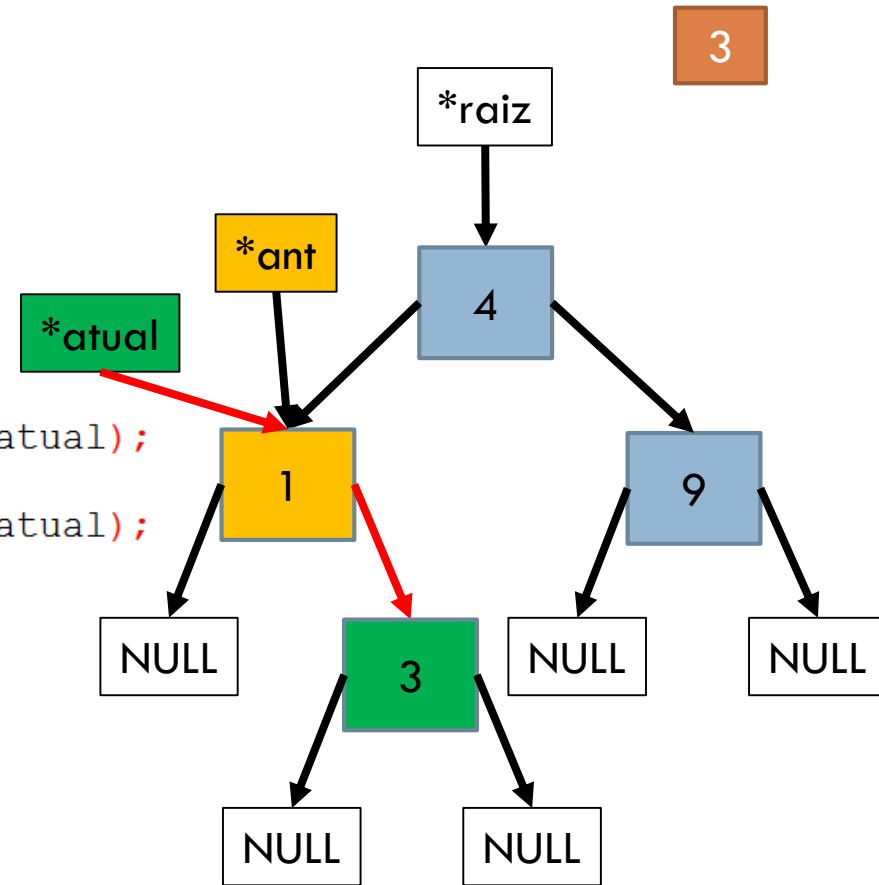
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

148

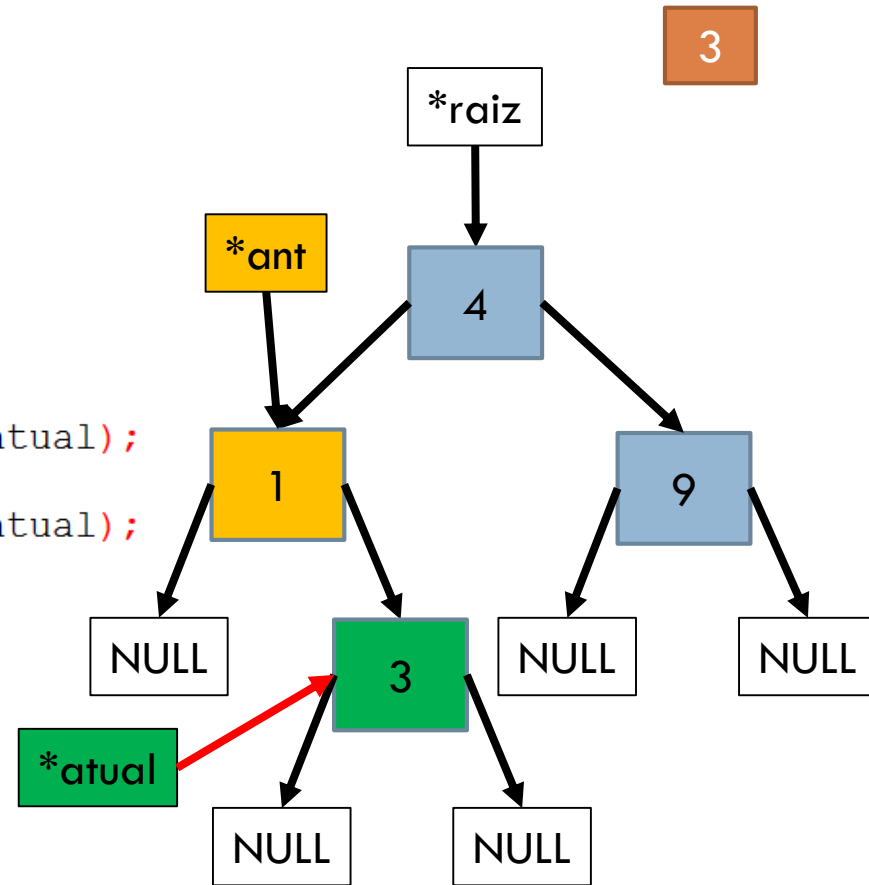
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

149

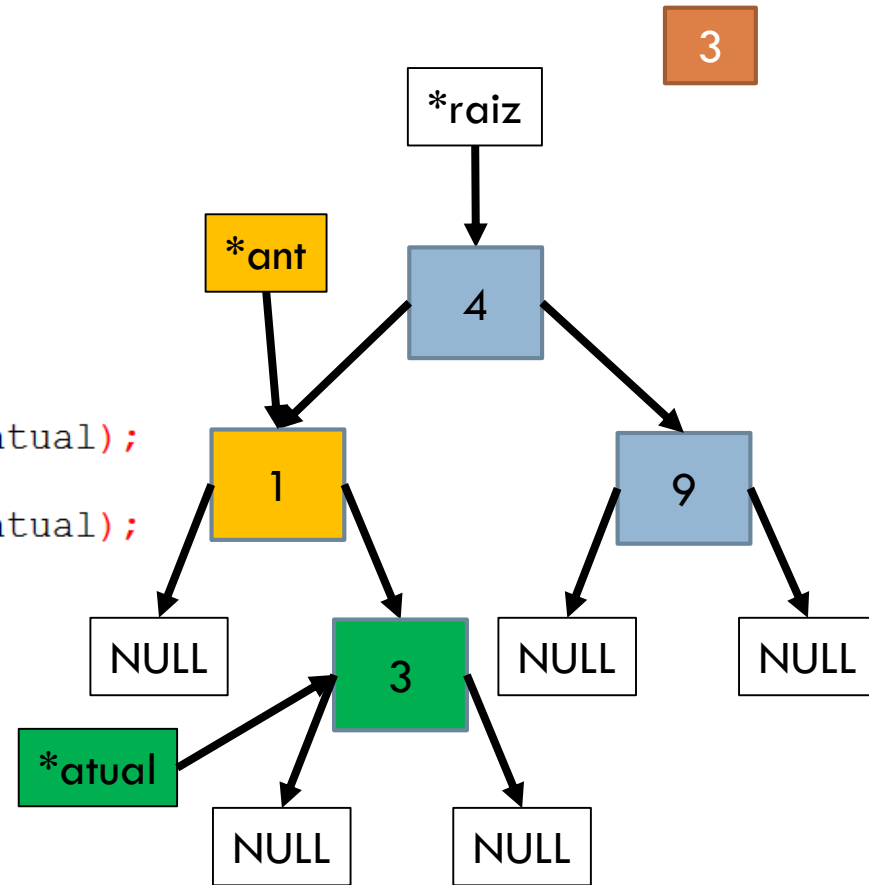
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

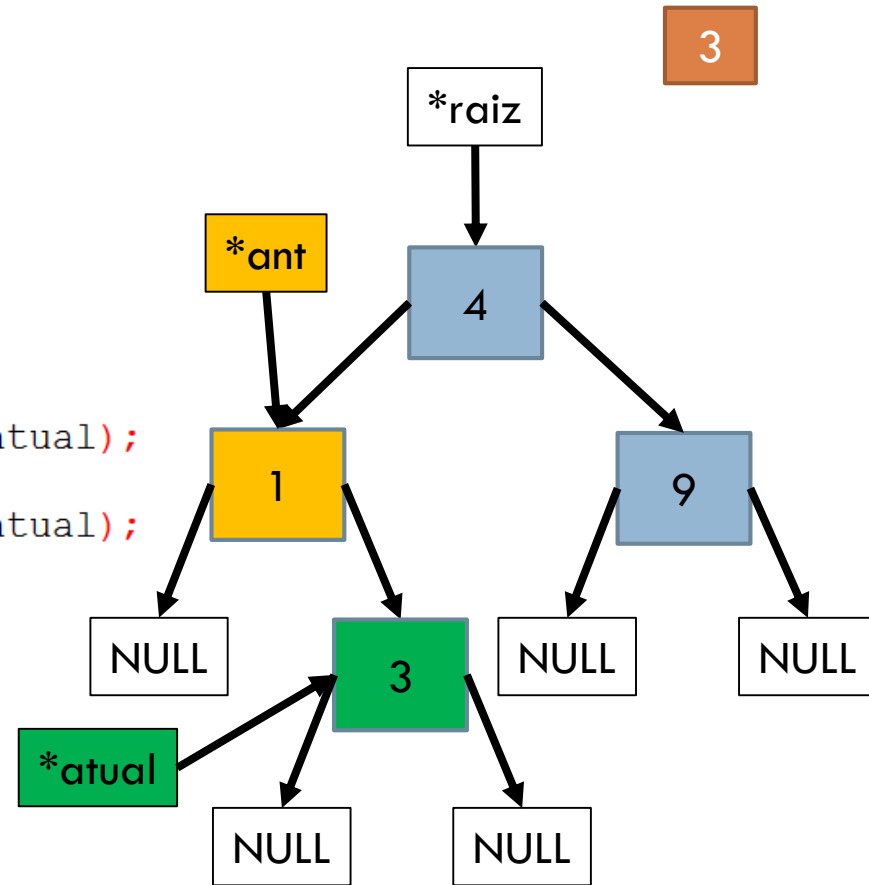
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    → while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

151

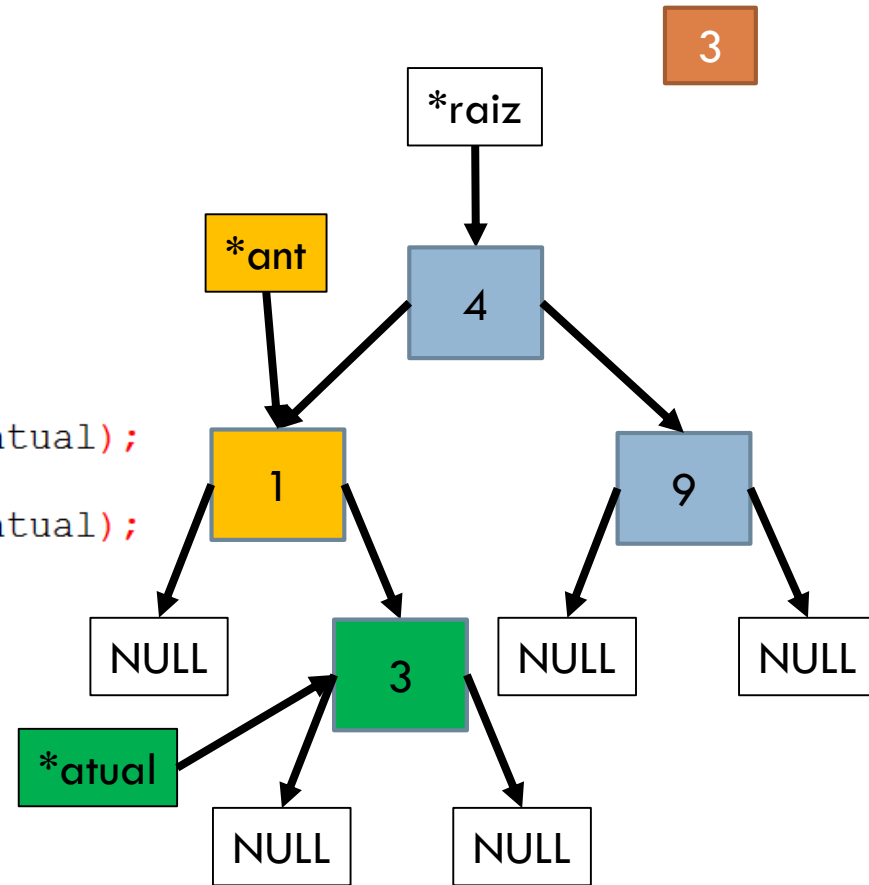
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        → if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

152

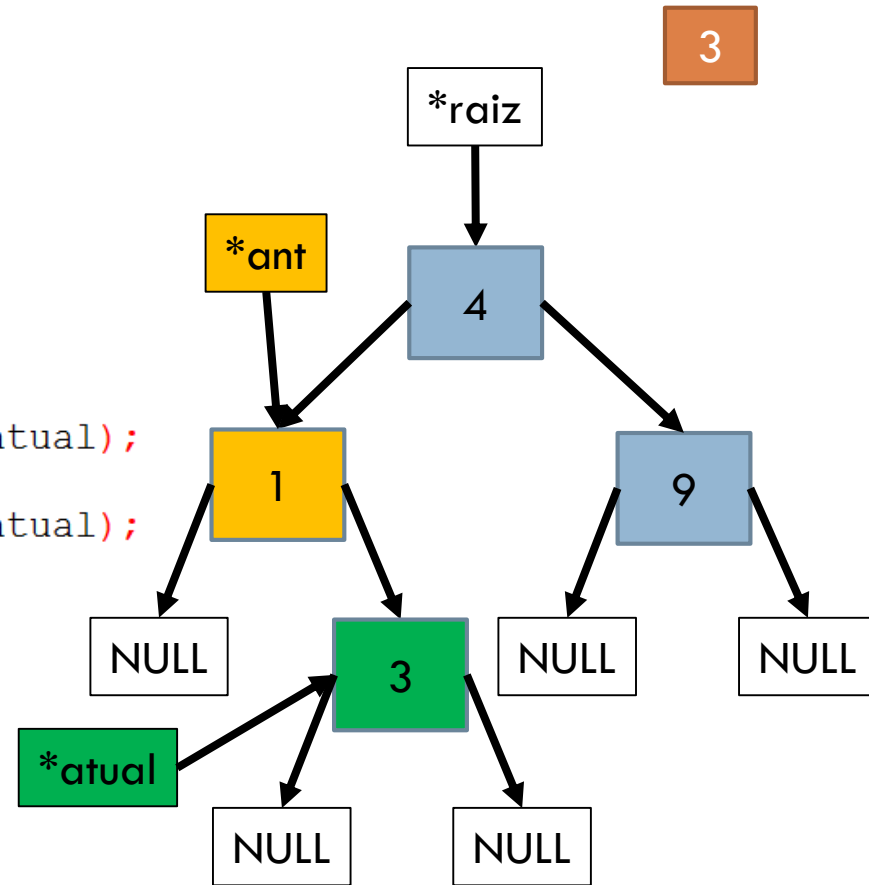
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            → if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

153

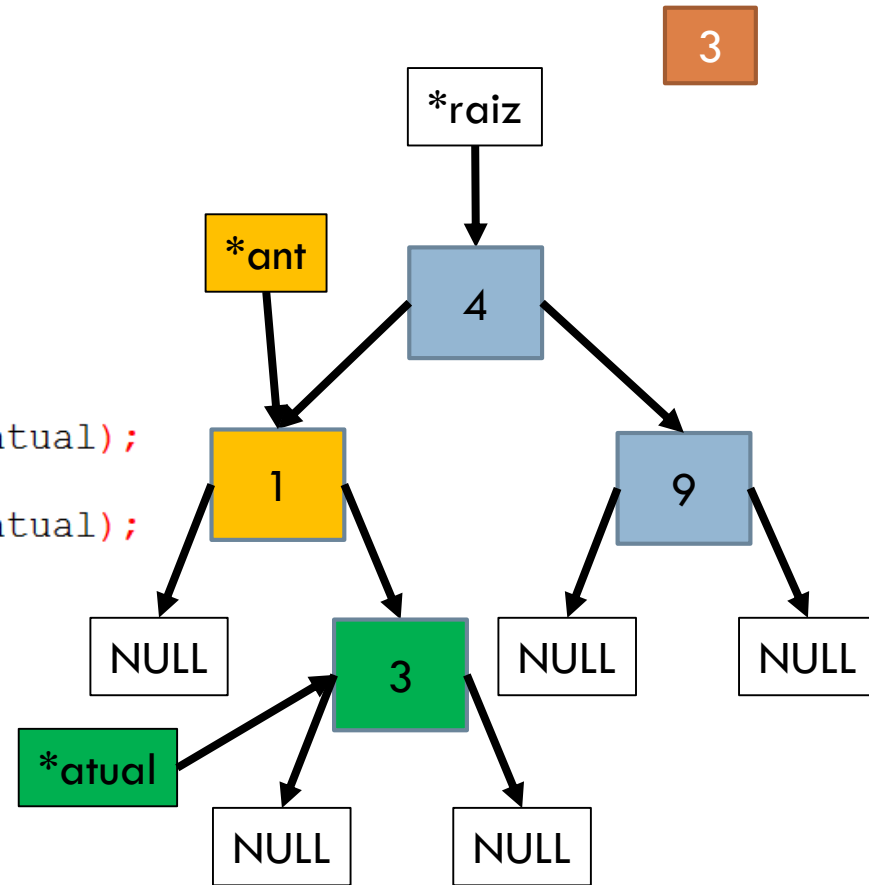
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            → else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

154

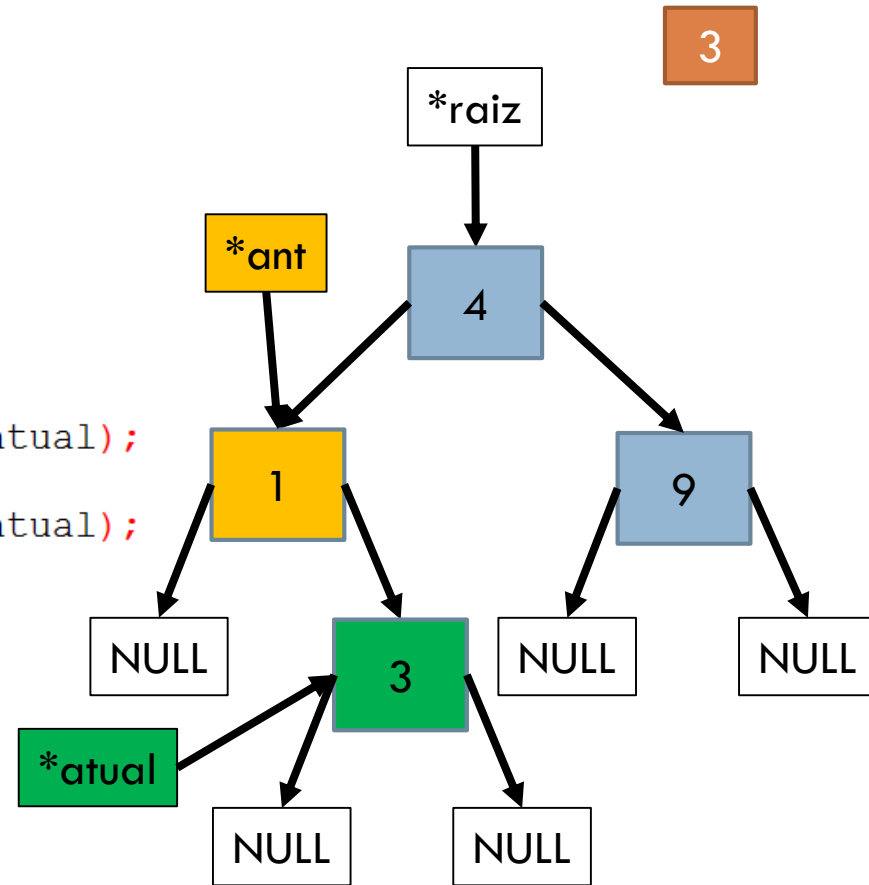
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

155

```
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;
}
```

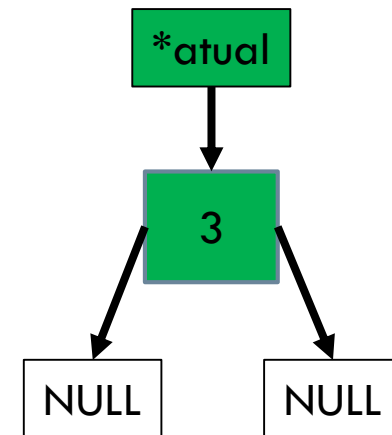


Remoção em ABB

156

```
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;
    }

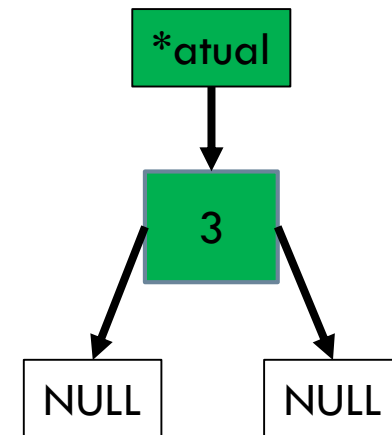
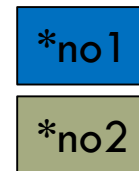
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

157

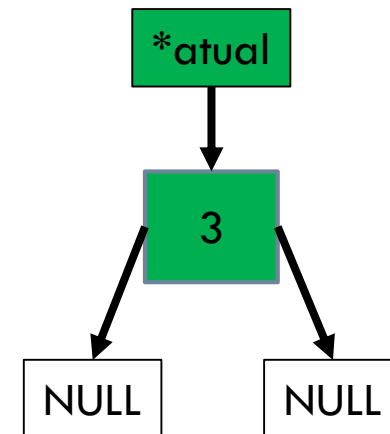
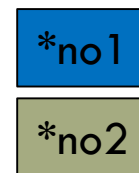
```
struct NO* remove_atual(struct NO* atual) {  
    → struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 =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



Remoção em ABB

158

```
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;  
}
```

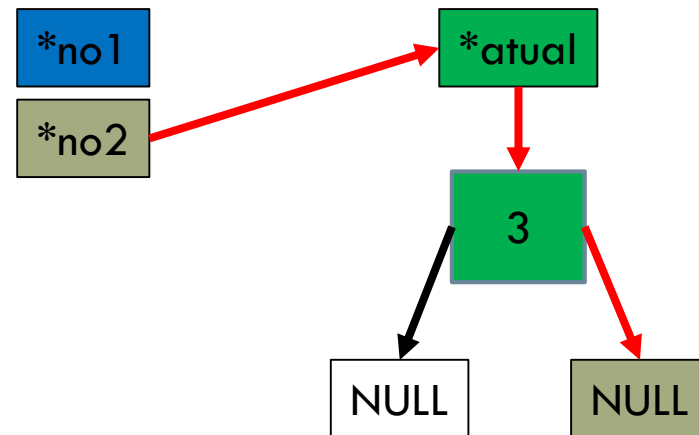


Remoção em ABB

159

```
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;
    }

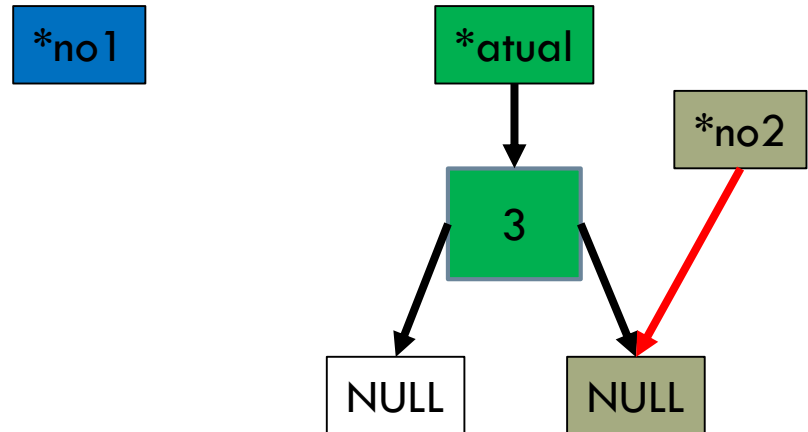
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

160

```
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;  
}
```



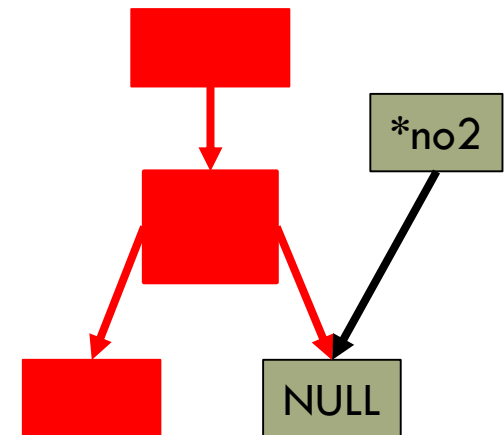
Remoção em ABB

161

```
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

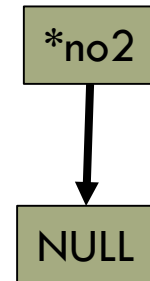


Remoção em ABB

162

```
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;  
}
```

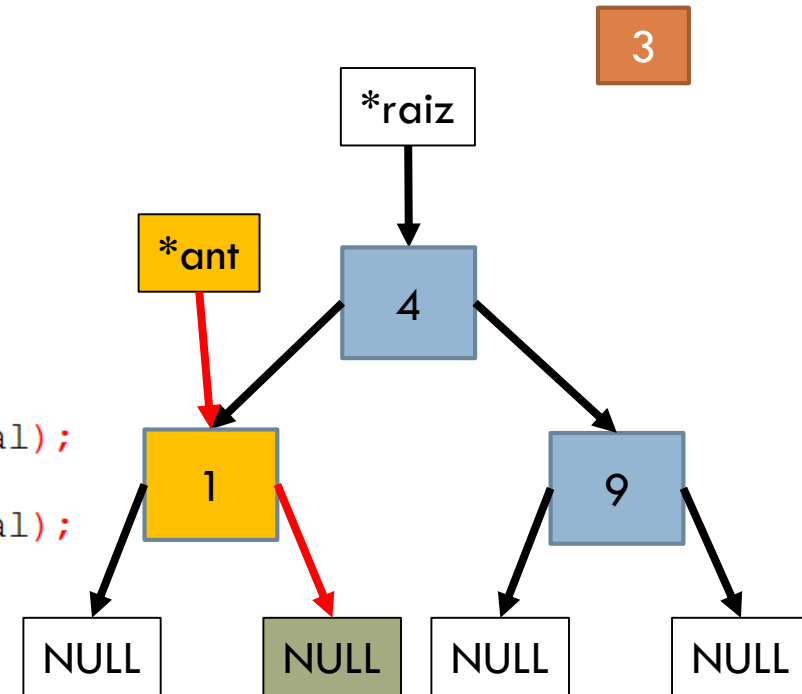
*no1



Remoção em ABB

163

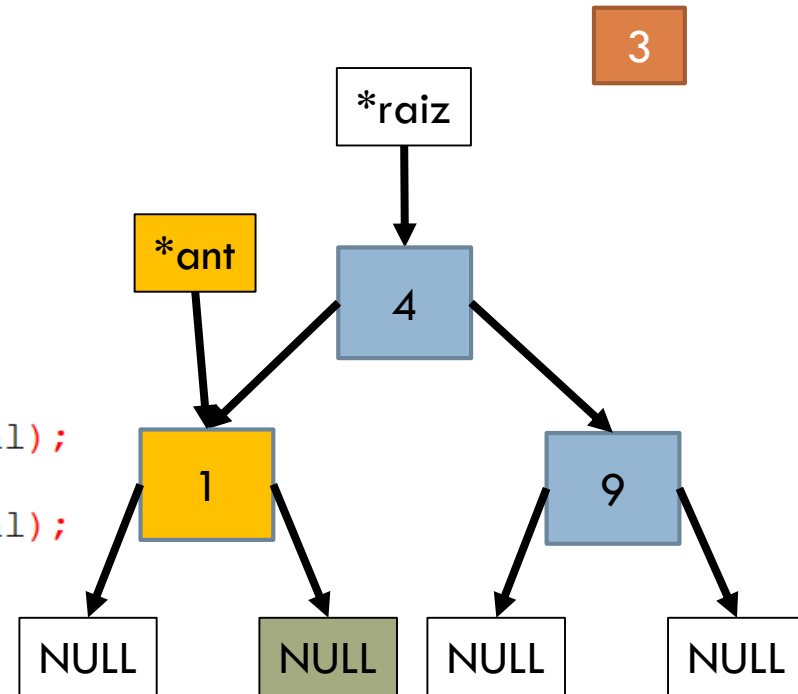
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

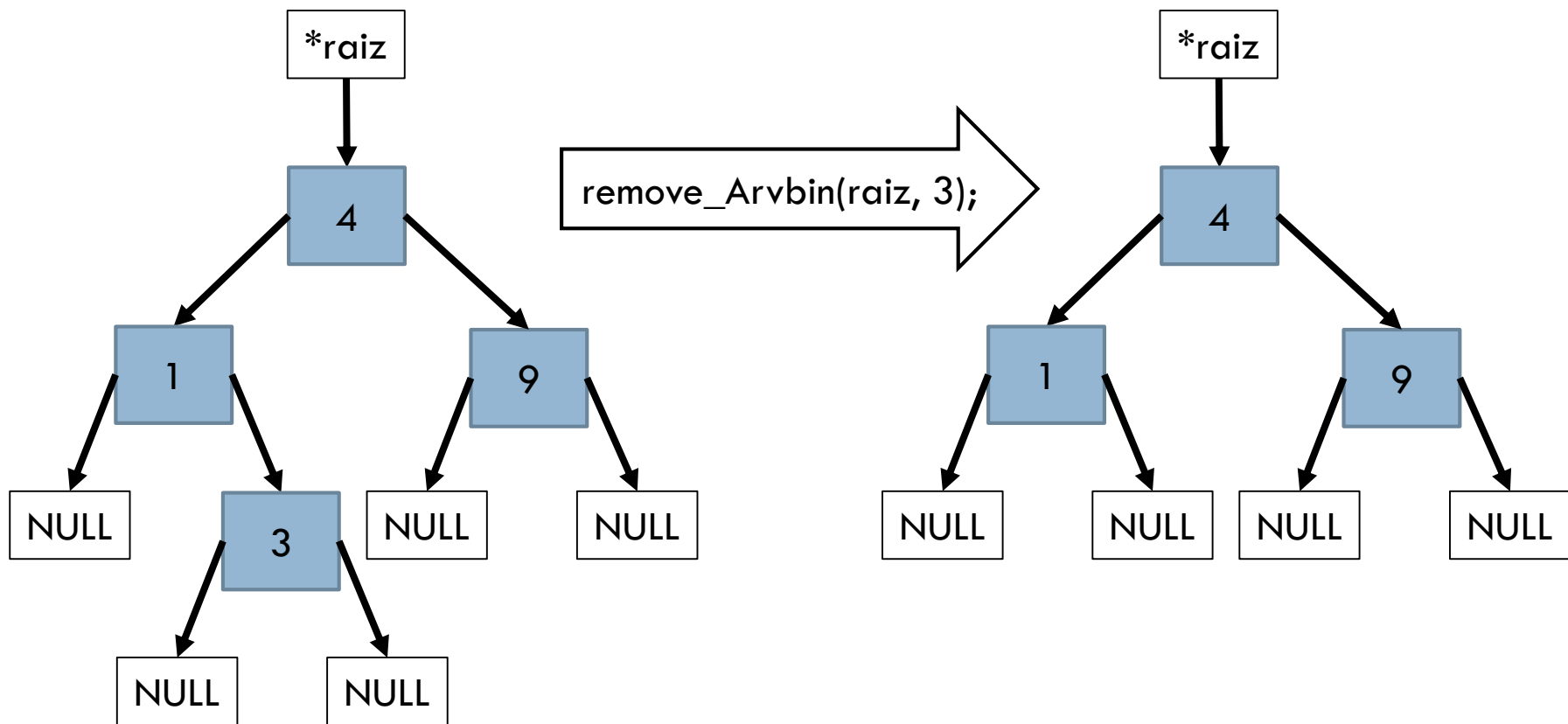
164

```
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

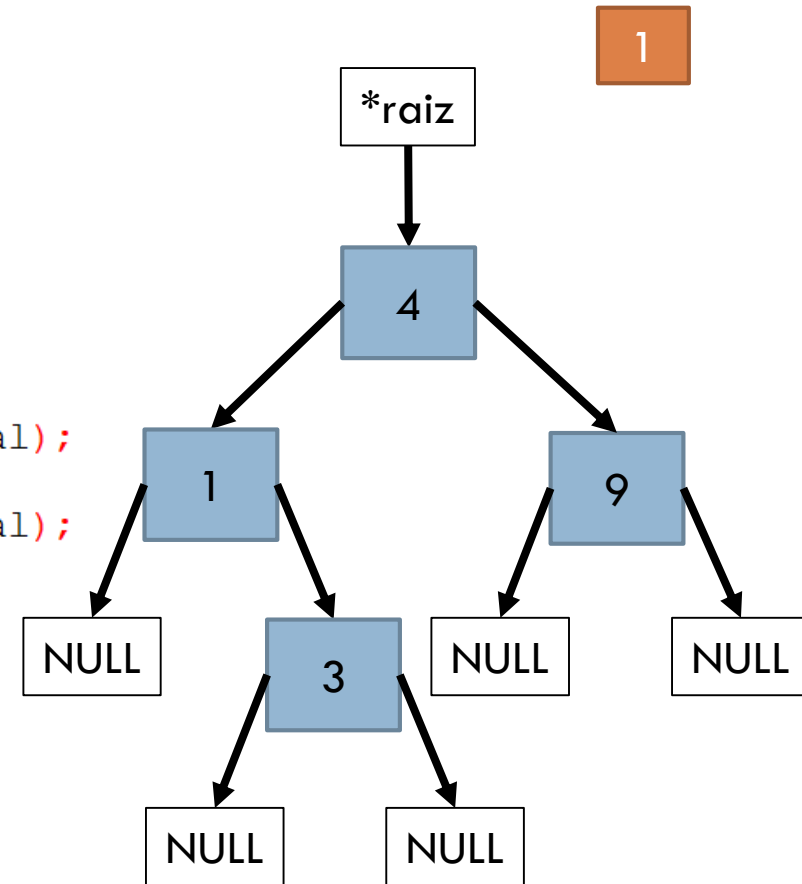
165



Remoção em ABB

166

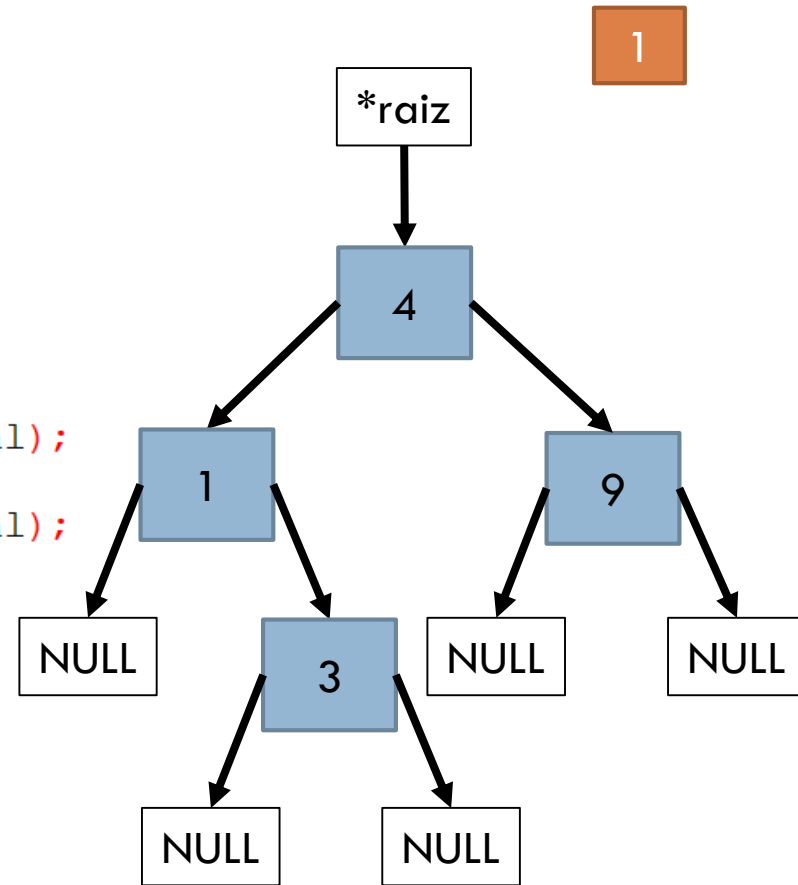
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

167

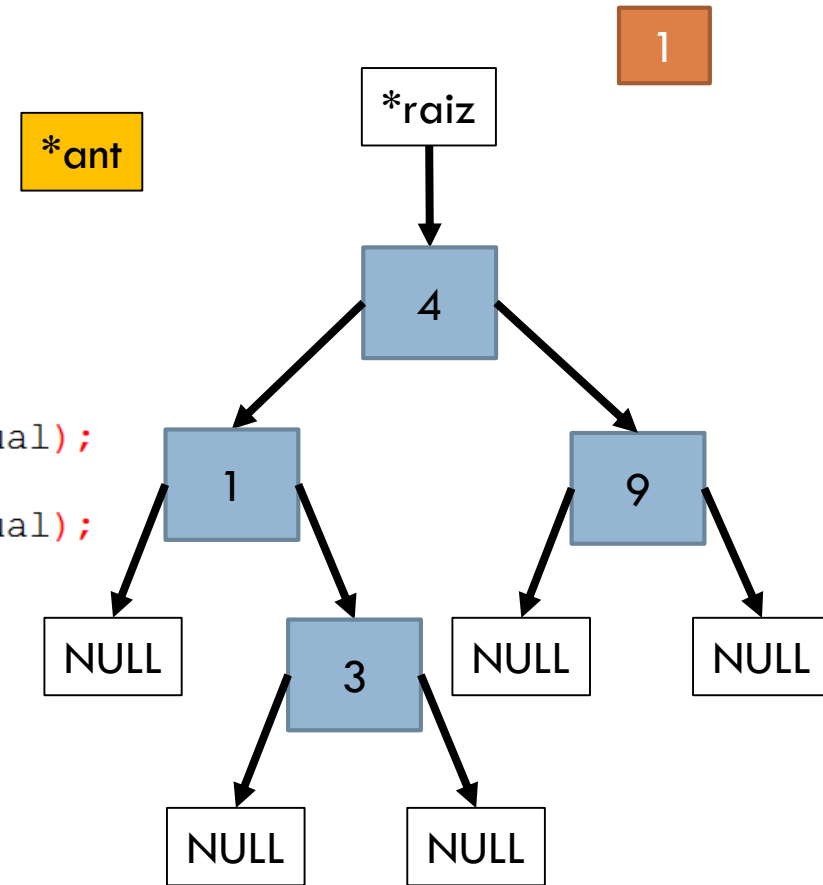
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    → if(raiz == NULL)  
        return 0;  
    struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    while(atual != NULL) {  
        if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

168

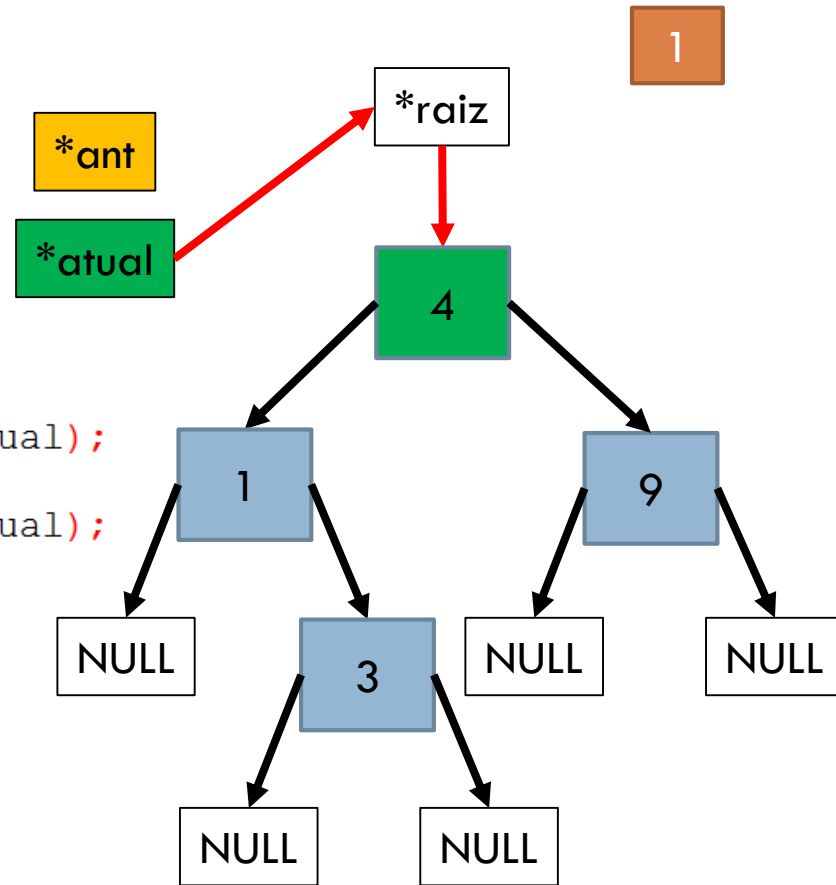
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    → struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

169

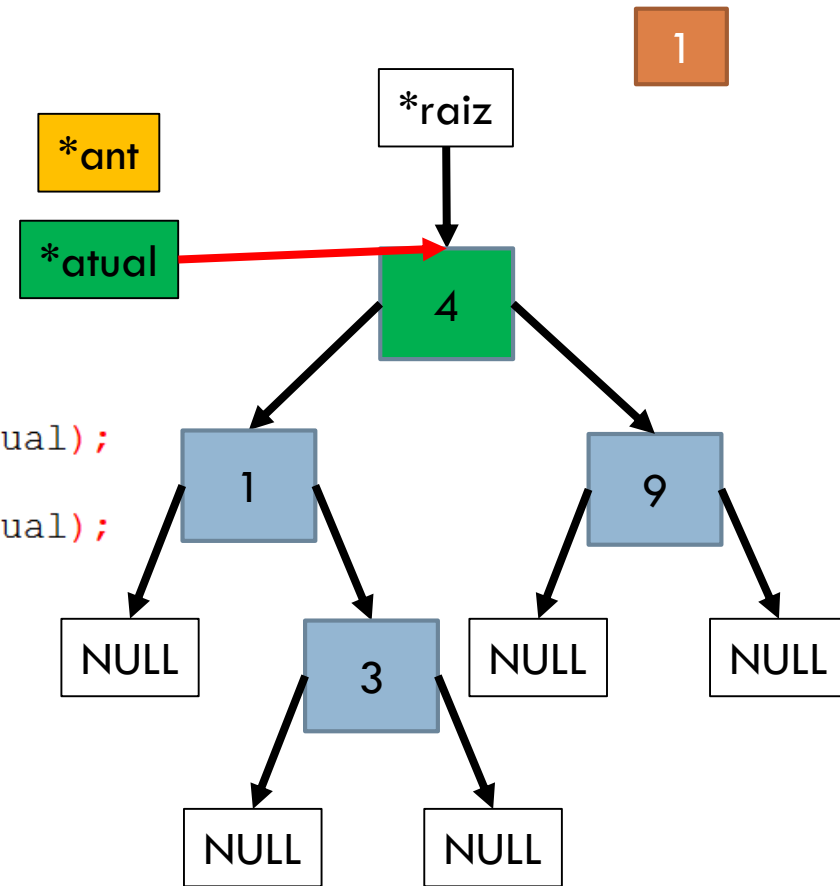
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

170

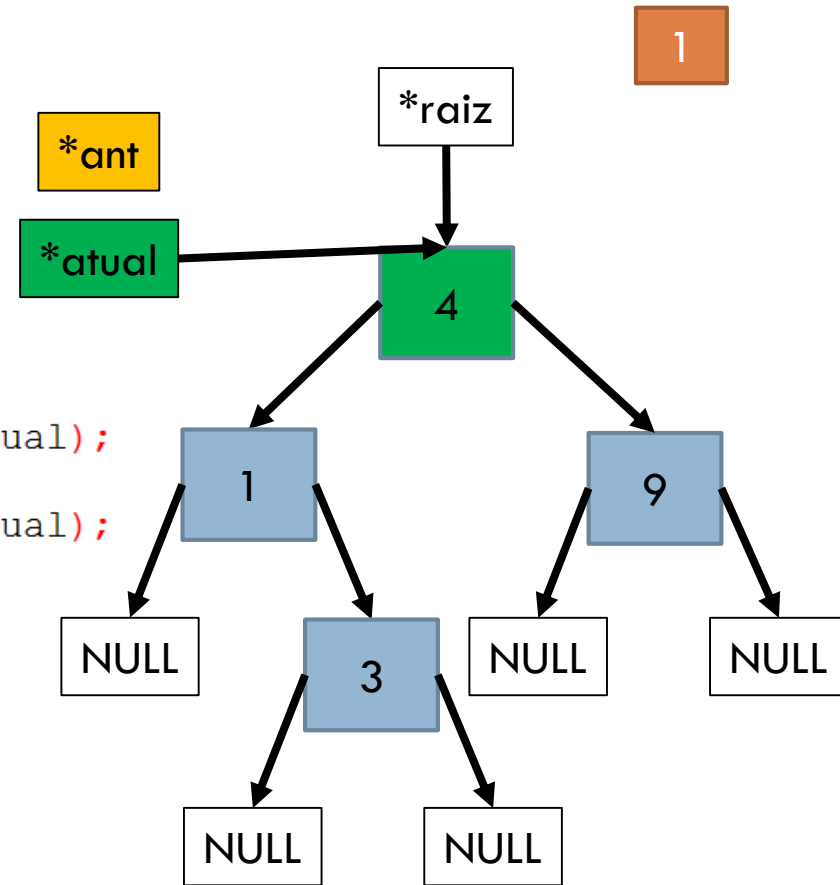
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

171

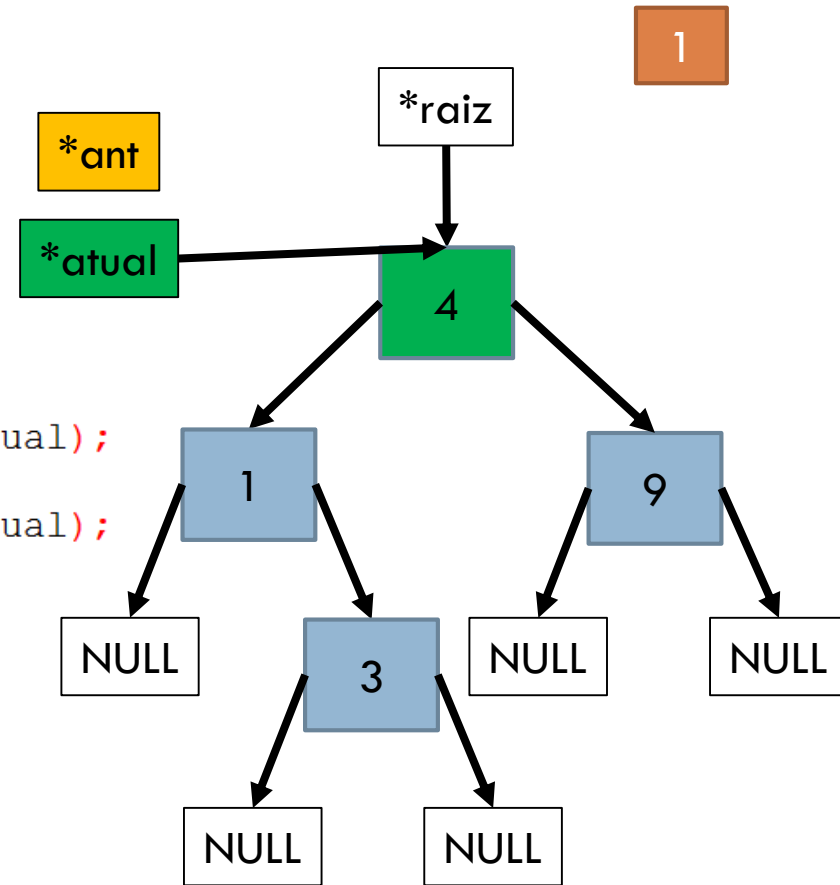
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    → while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

172

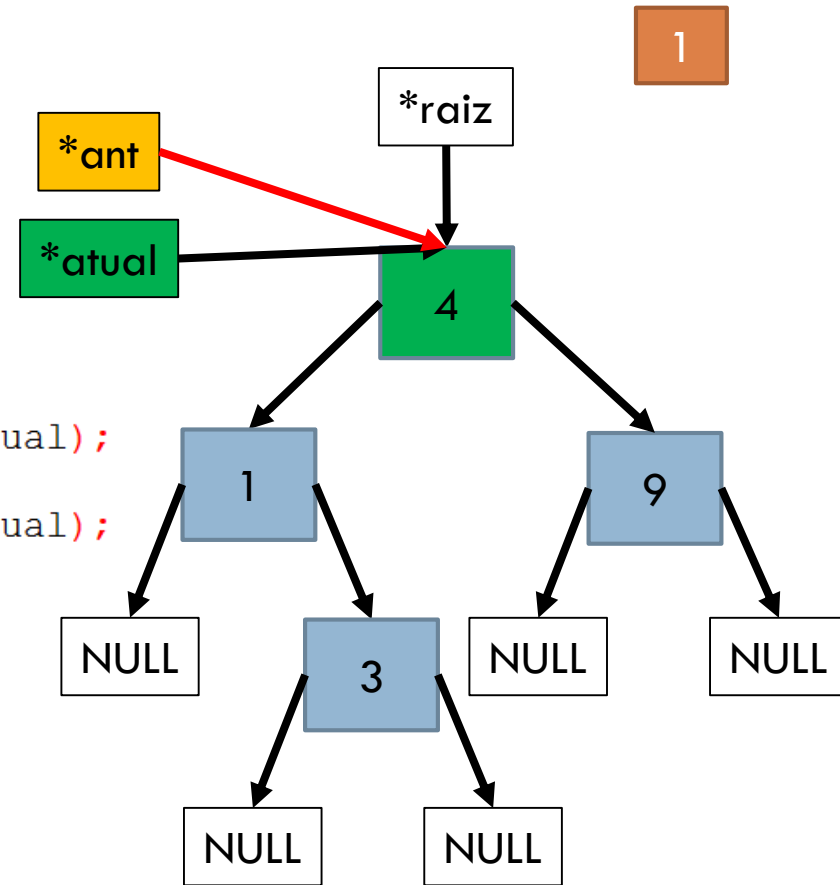
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        → if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

173

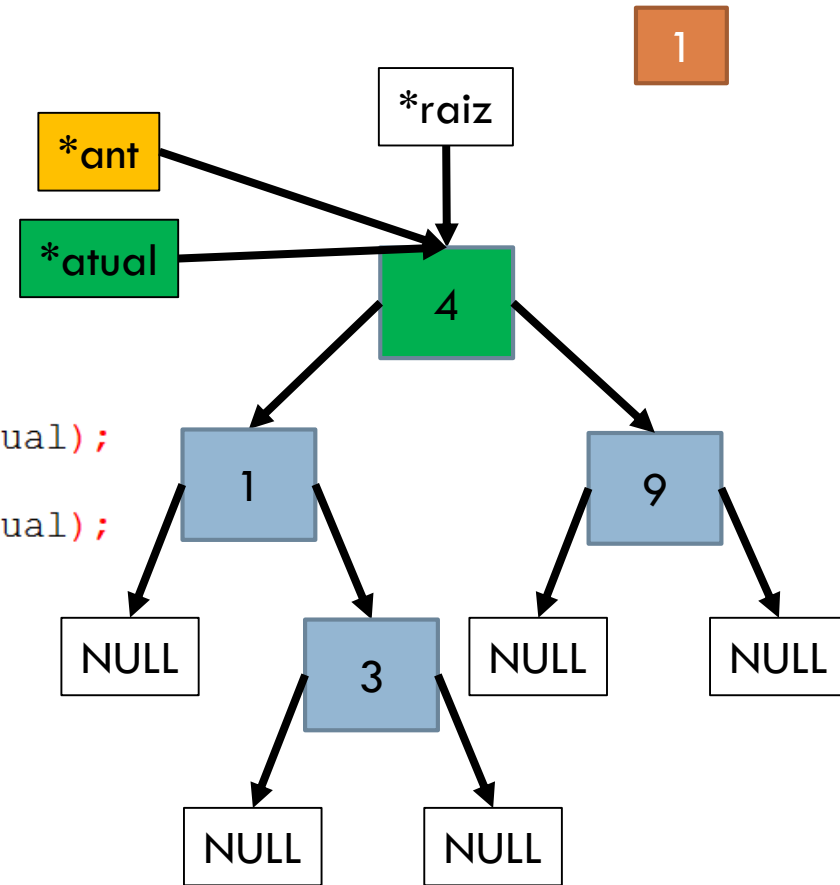
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

174

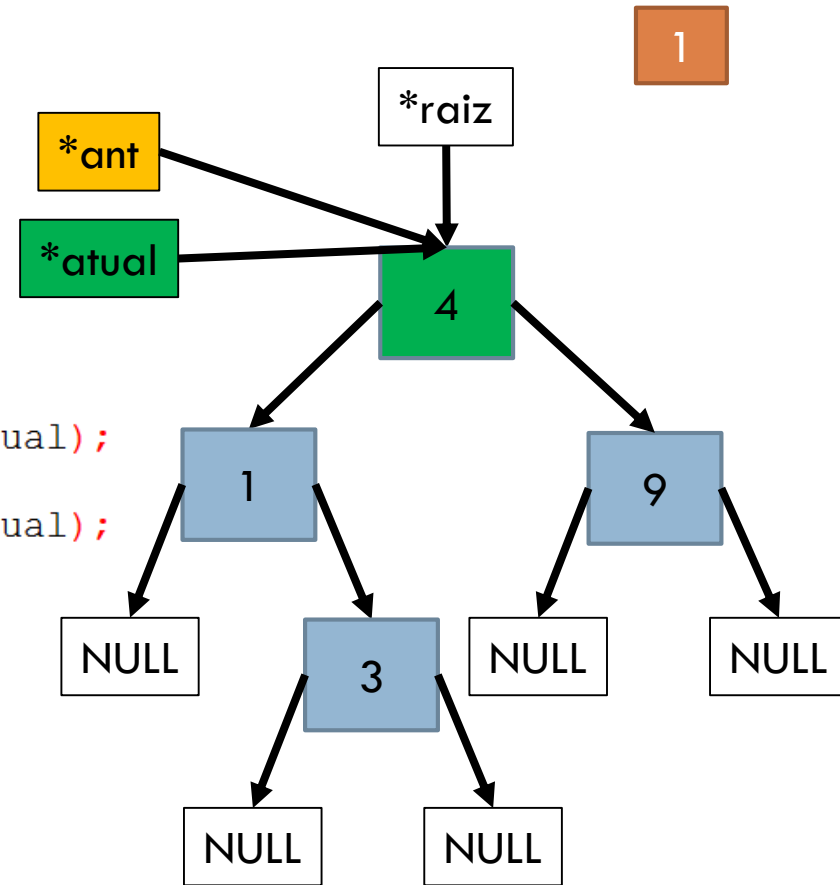
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

175

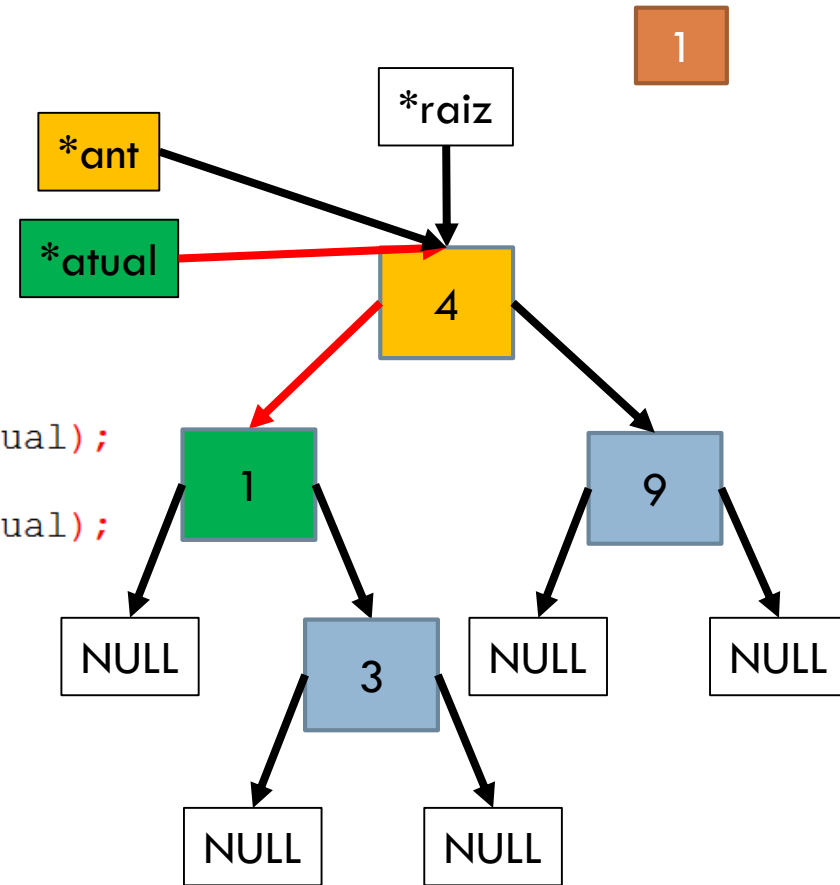
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

176

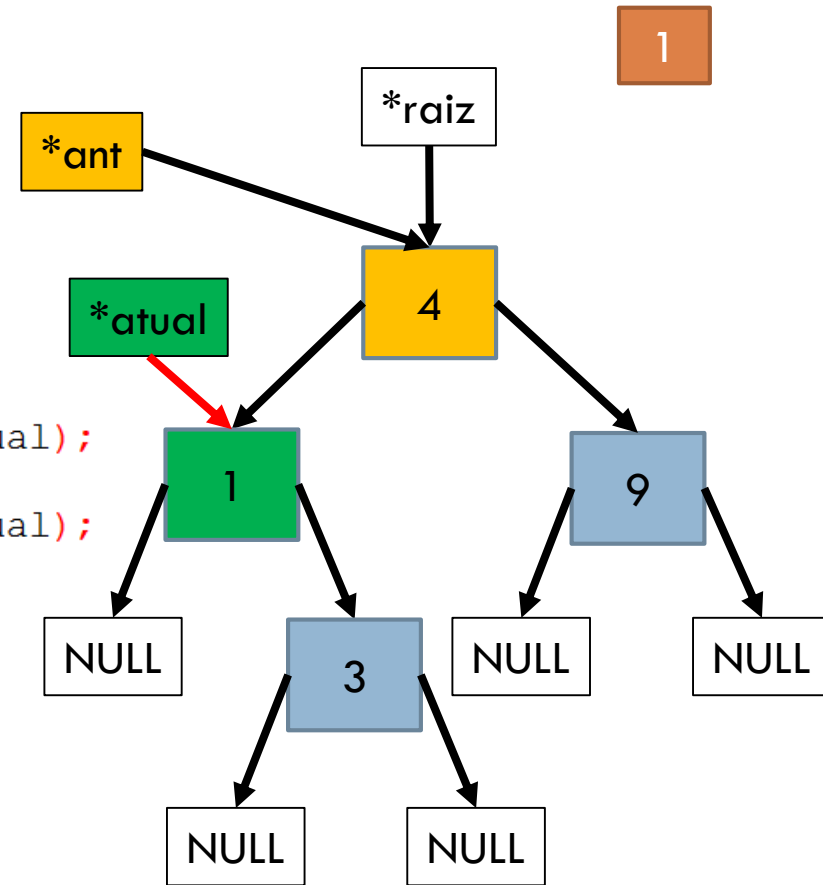
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

177

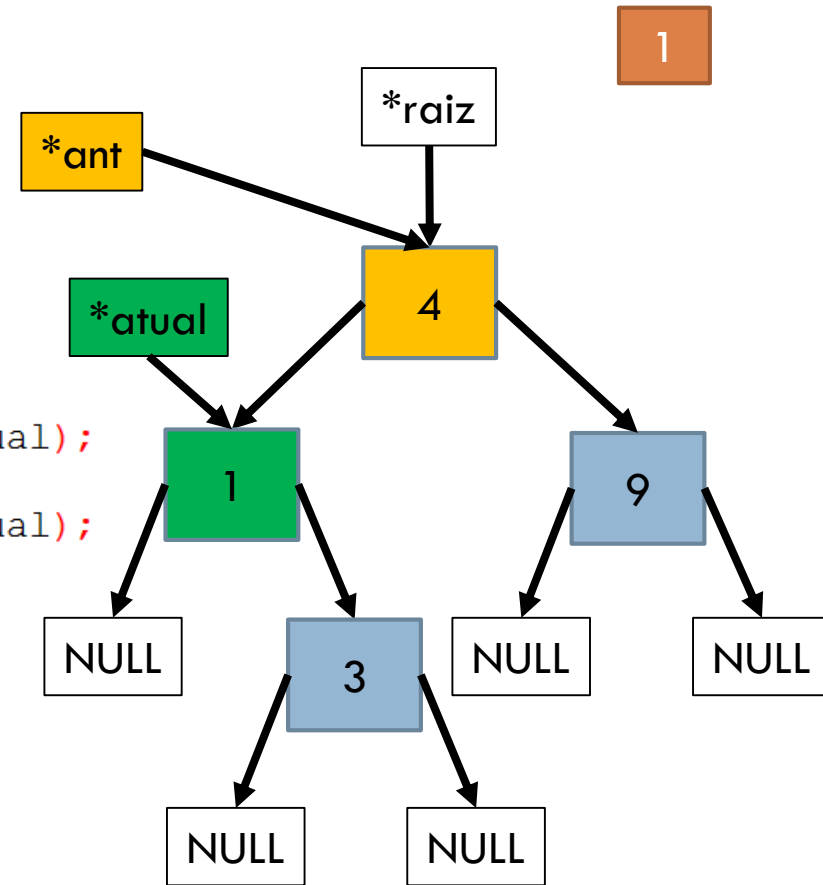
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

178

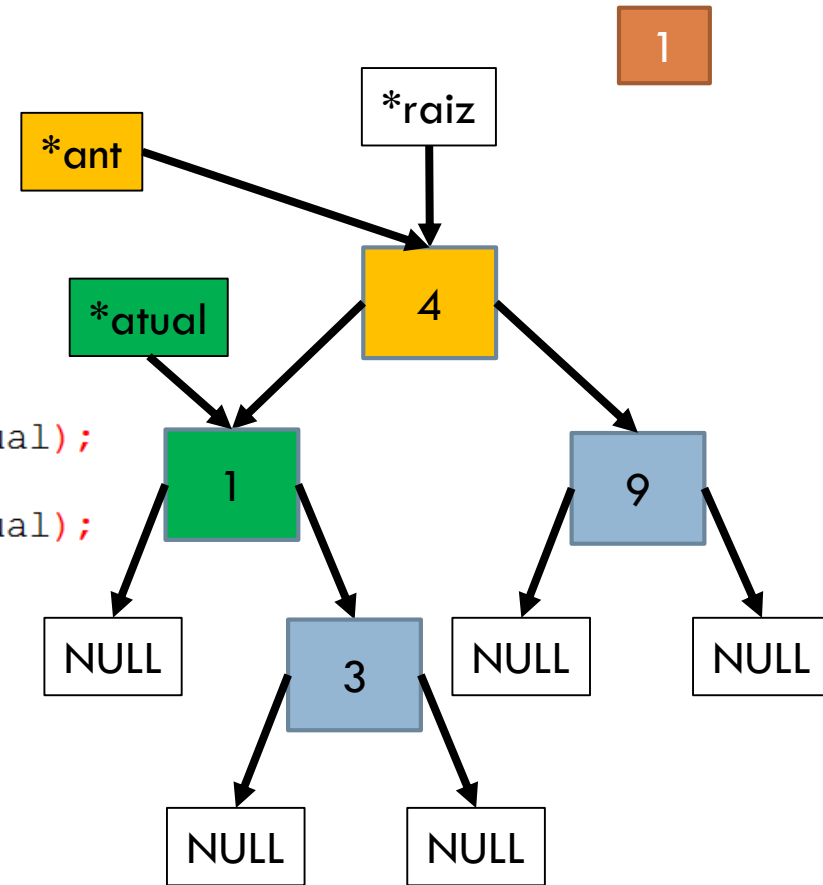
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    → while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

179

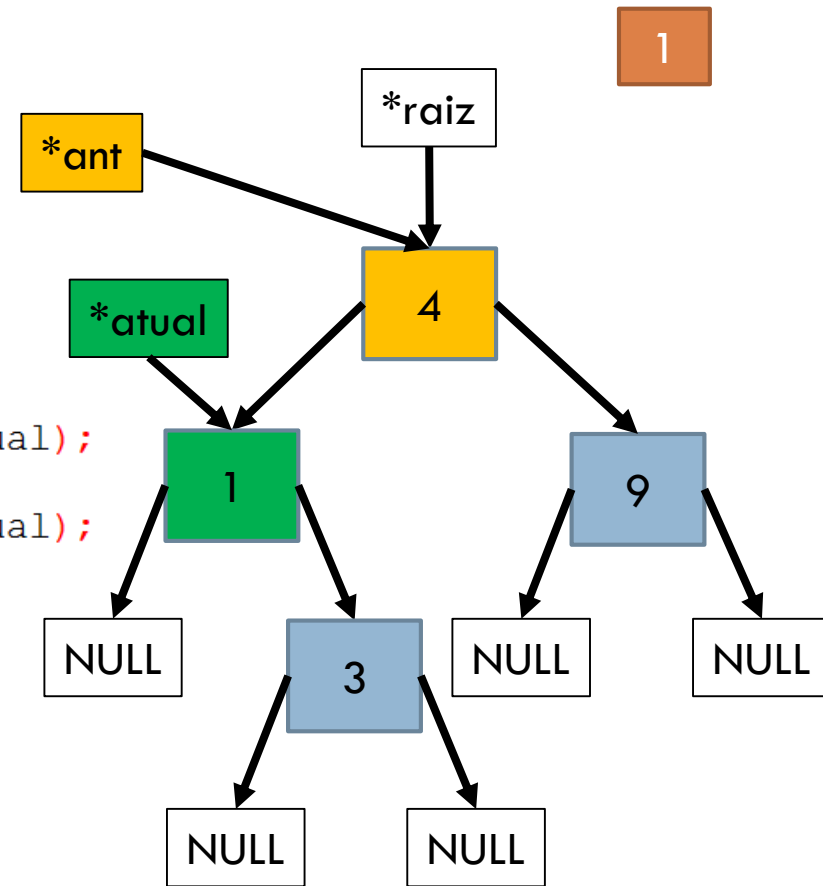
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        → if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

180

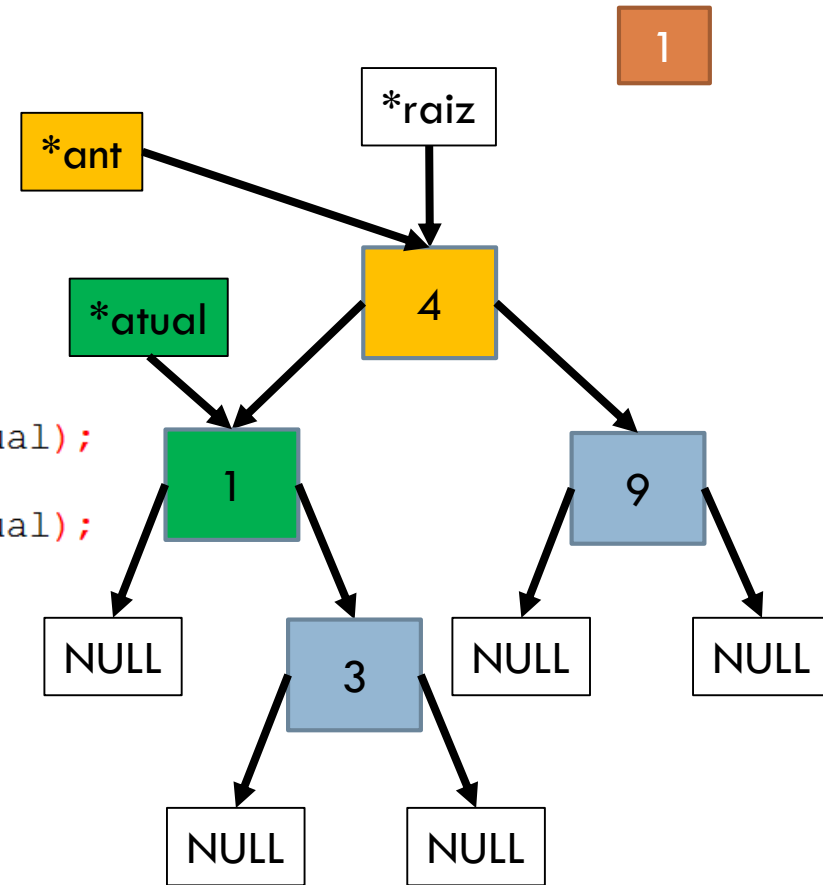
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            → if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

181

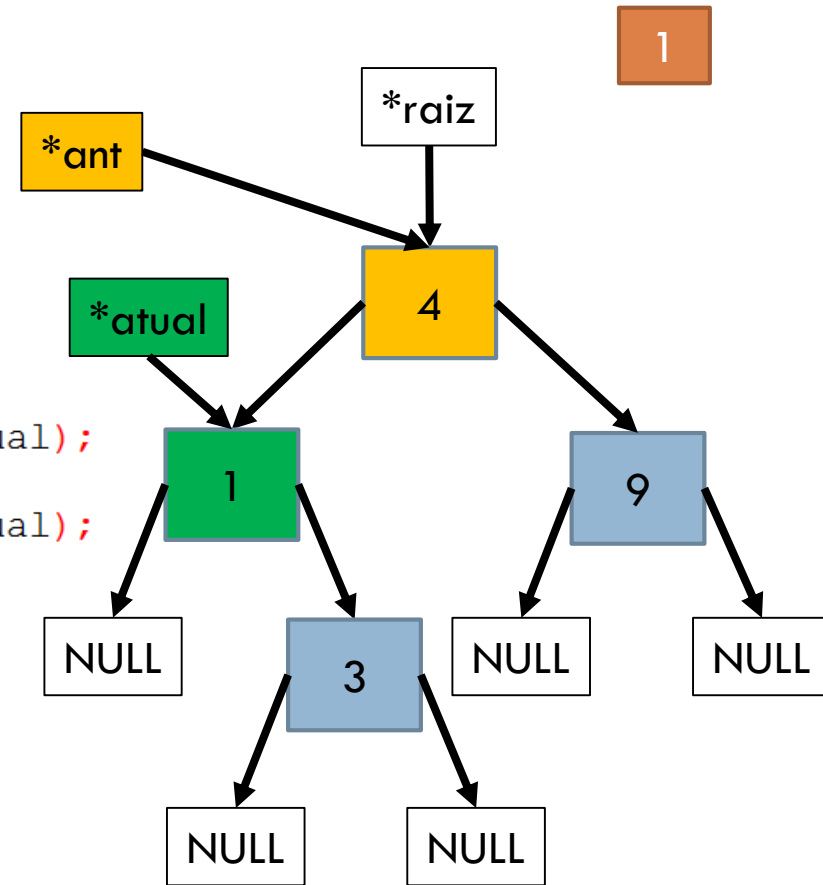
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            → else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

182

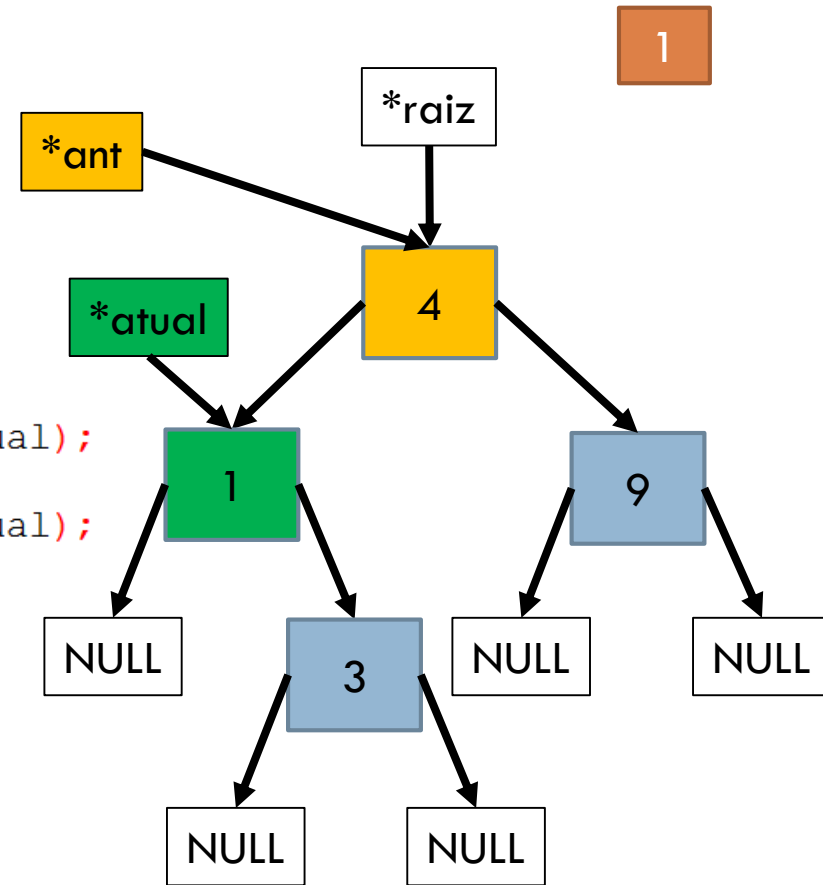
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

183

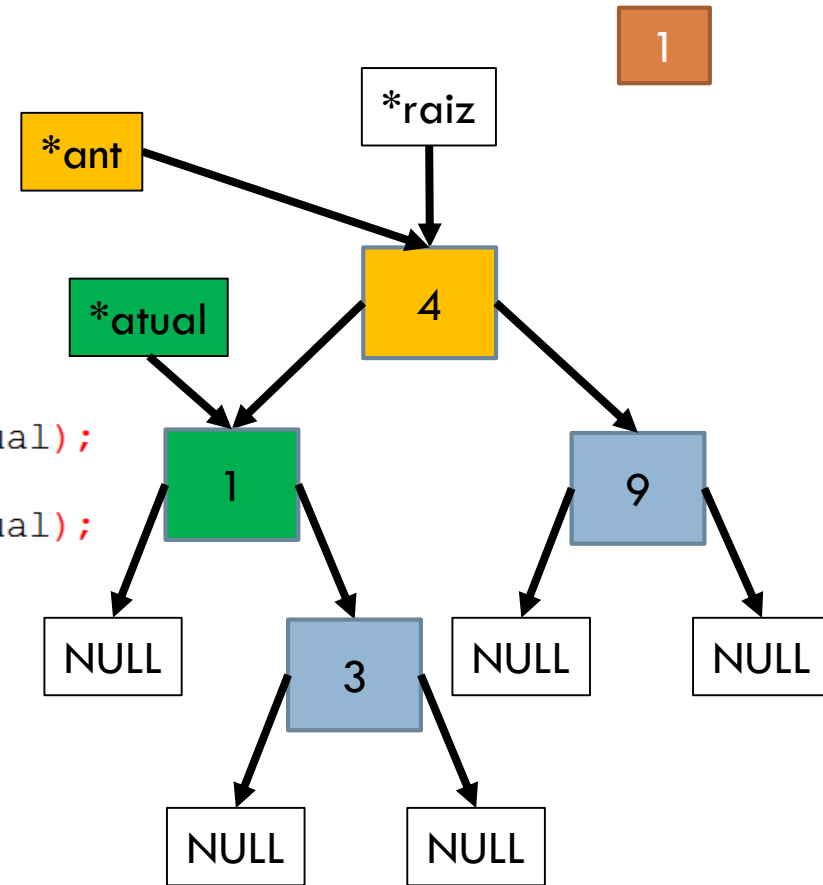
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

184

```
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;
}
```

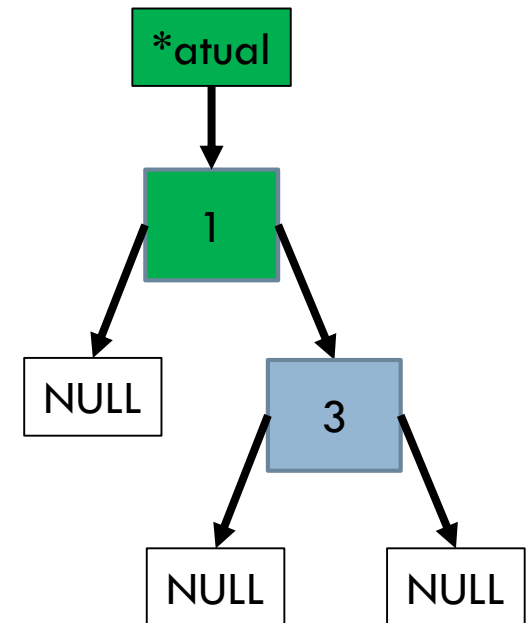


Remoção em ABB

185

```
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;
    }

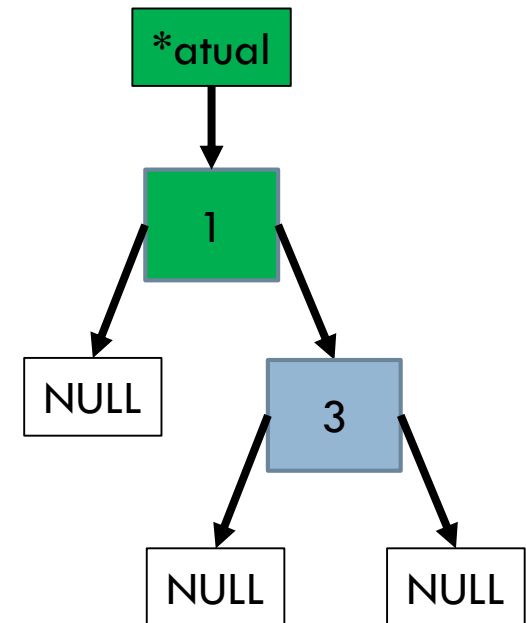
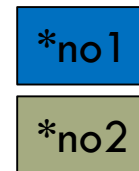
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

186

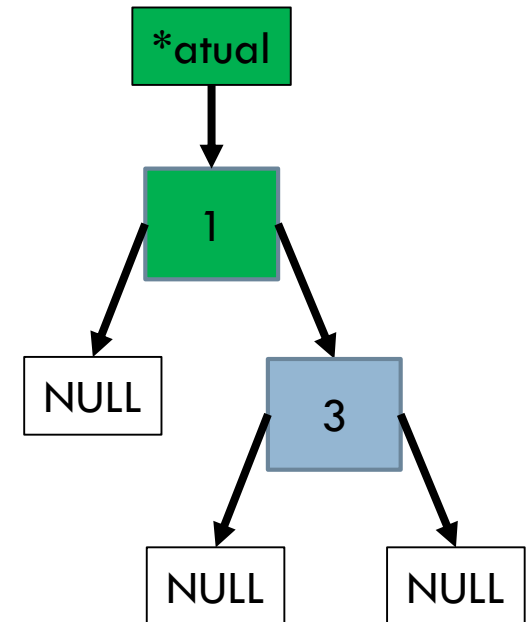
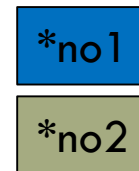
```
struct NO* remove_atual(struct NO* atual) {  
    → struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 =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



Remoção em ABB

187

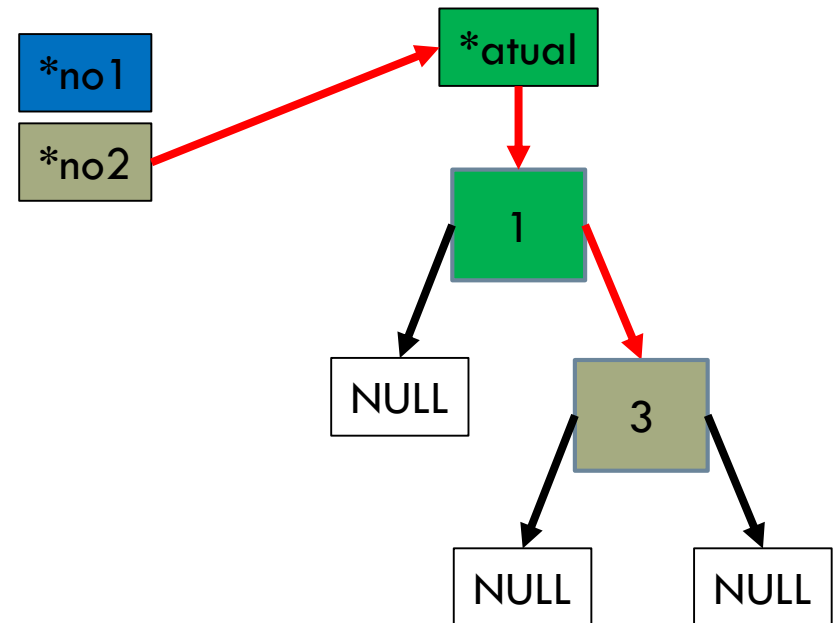
```
struct NO* remove_atual(struct NO* atual) {  
    struct NO *no1, *no2;  
    → if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 =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



Remoção em ABB

188

```
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;  
}
```

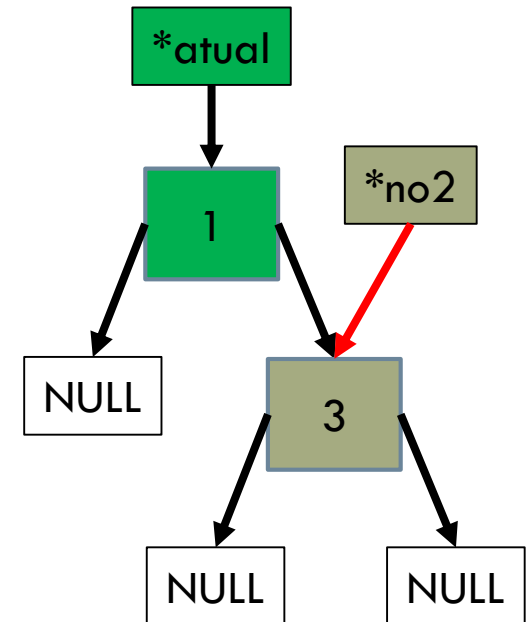


Remoção em ABB

189

```
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

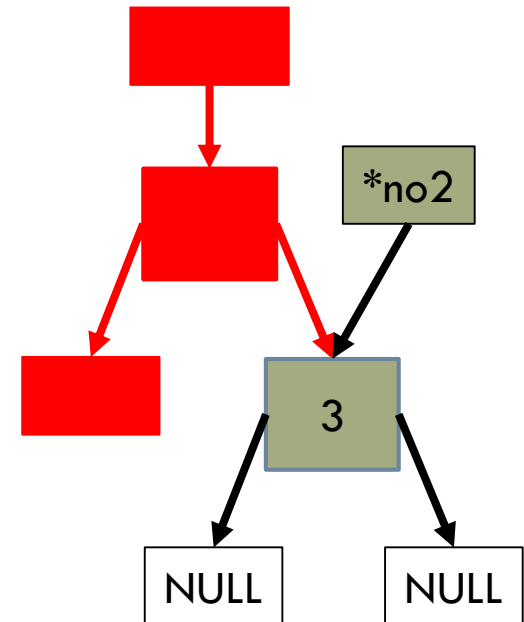


Remoção em ABB

190

```
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

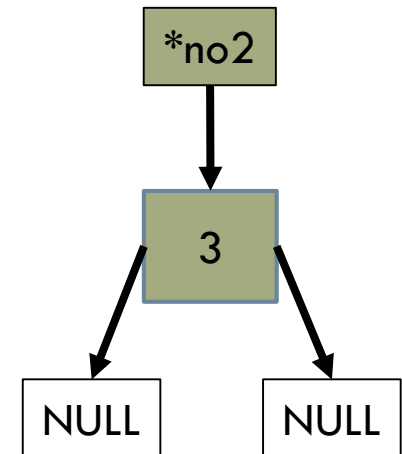


Remoção em ABB

191

```
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;  
}
```

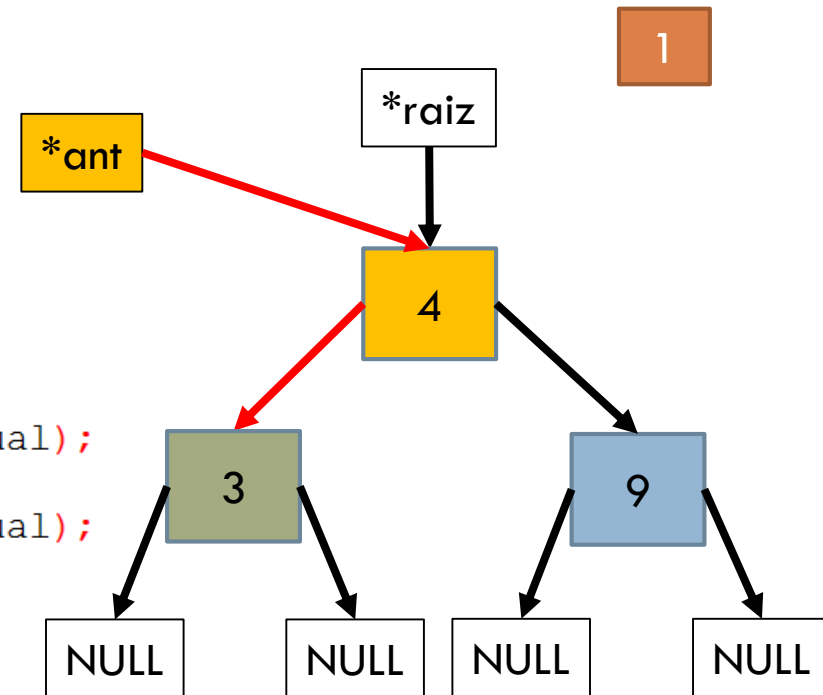
*no1



Remoção em ABB

192

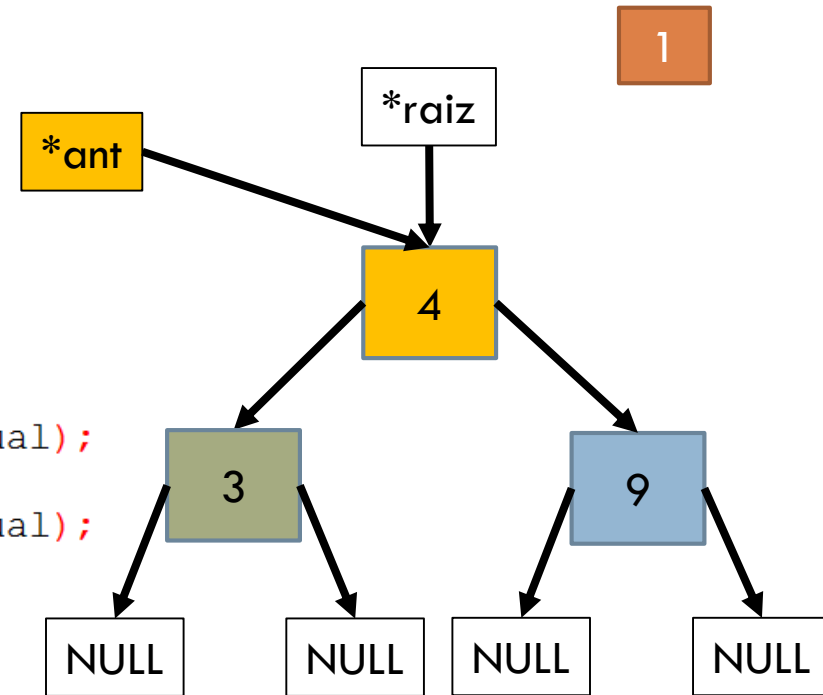
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

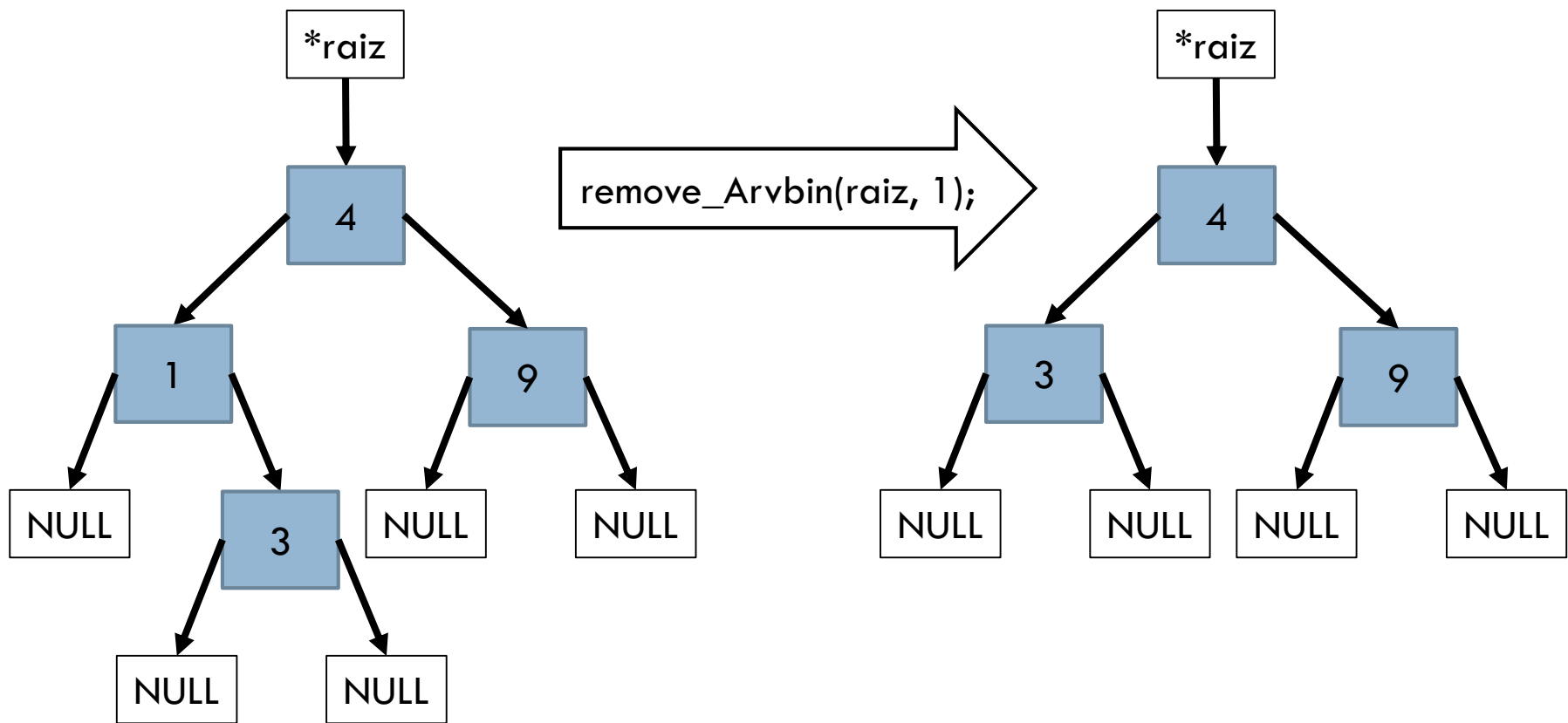
193

```
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

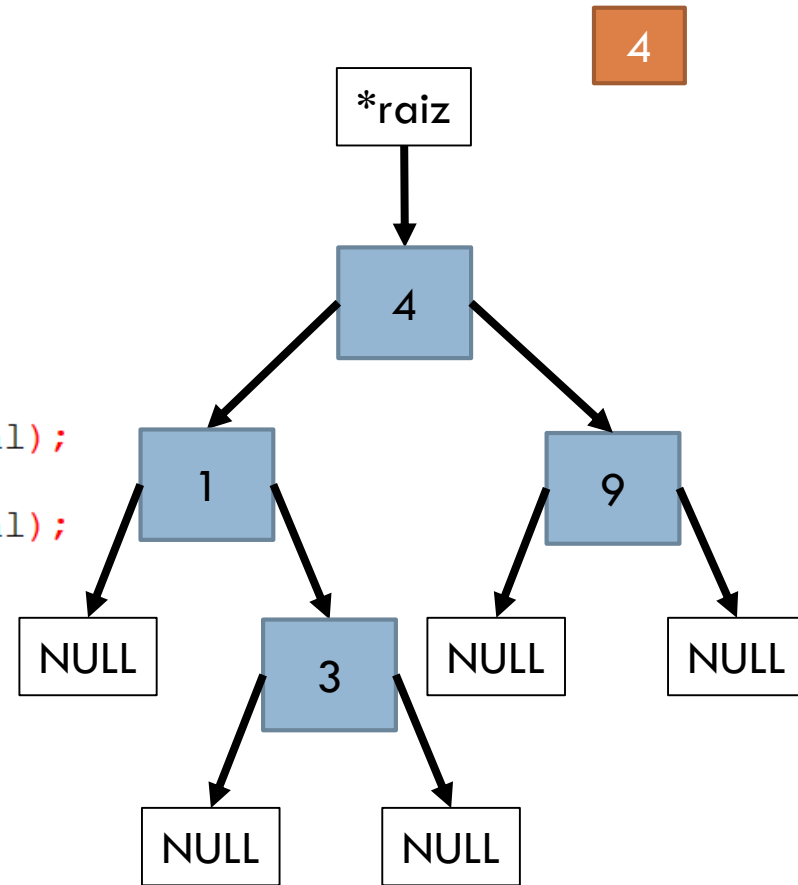
194



Remoção em ABB

195

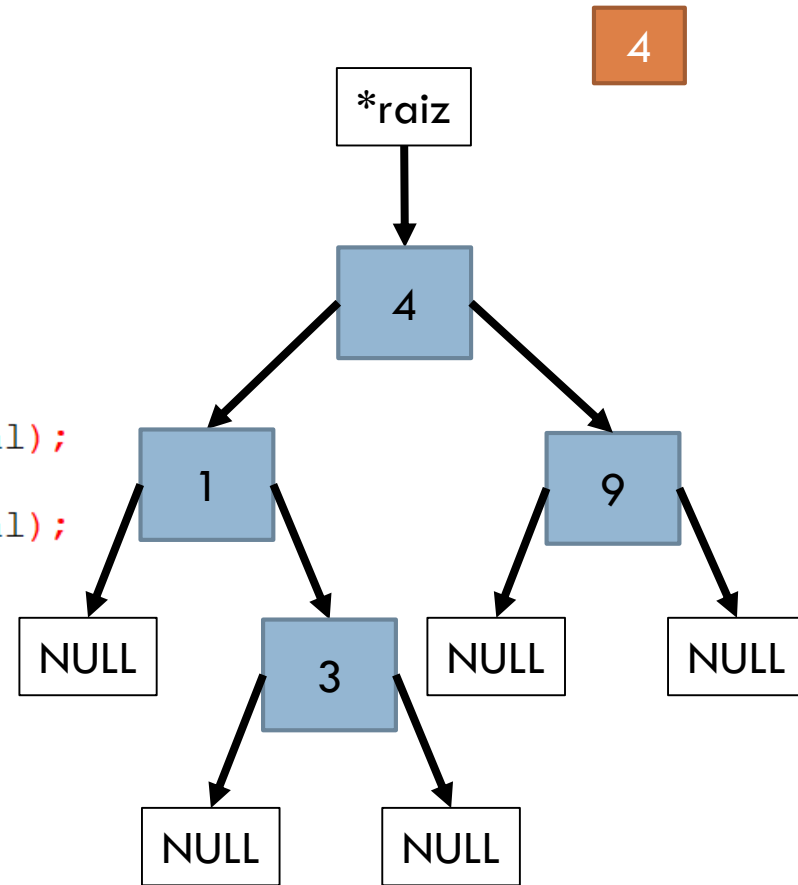
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else {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

196

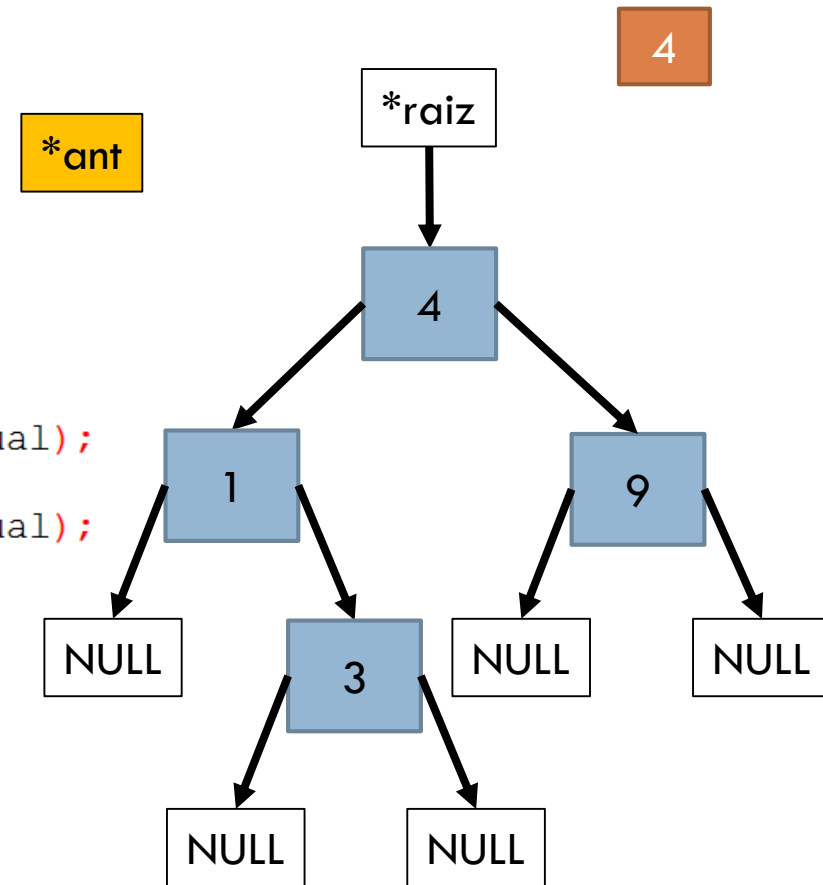
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    → if(raiz == NULL)  
        return 0;  
    struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    while(atual != NULL) {  
        if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

197

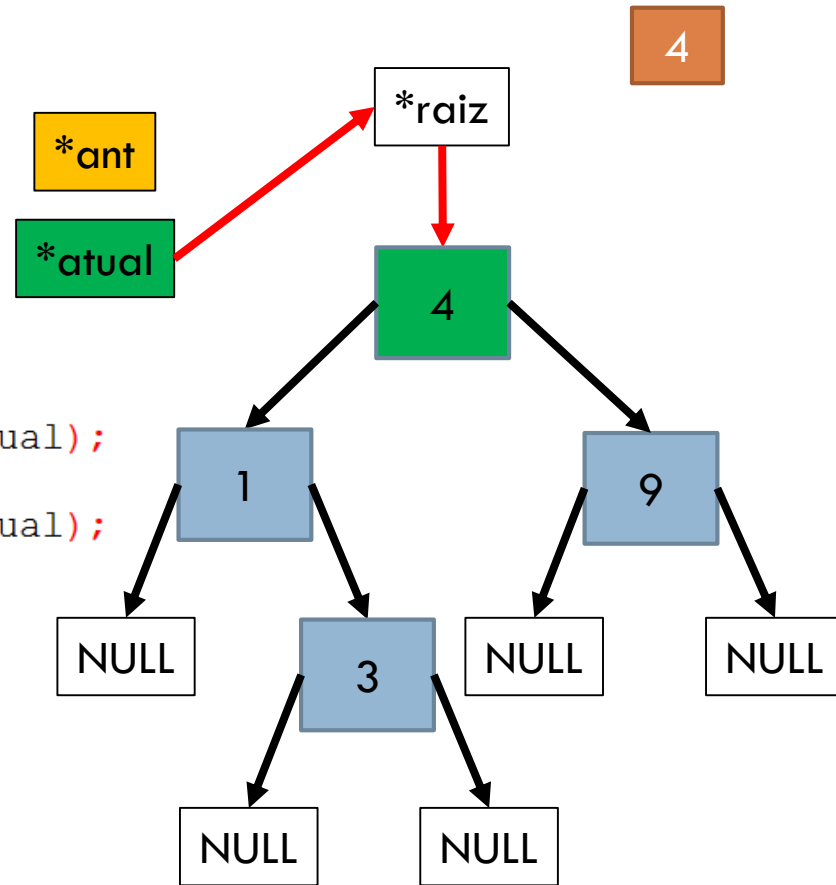
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    if(raiz == NULL)  
        return 0;  
    → struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    while(atual != NULL) {  
        if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

198

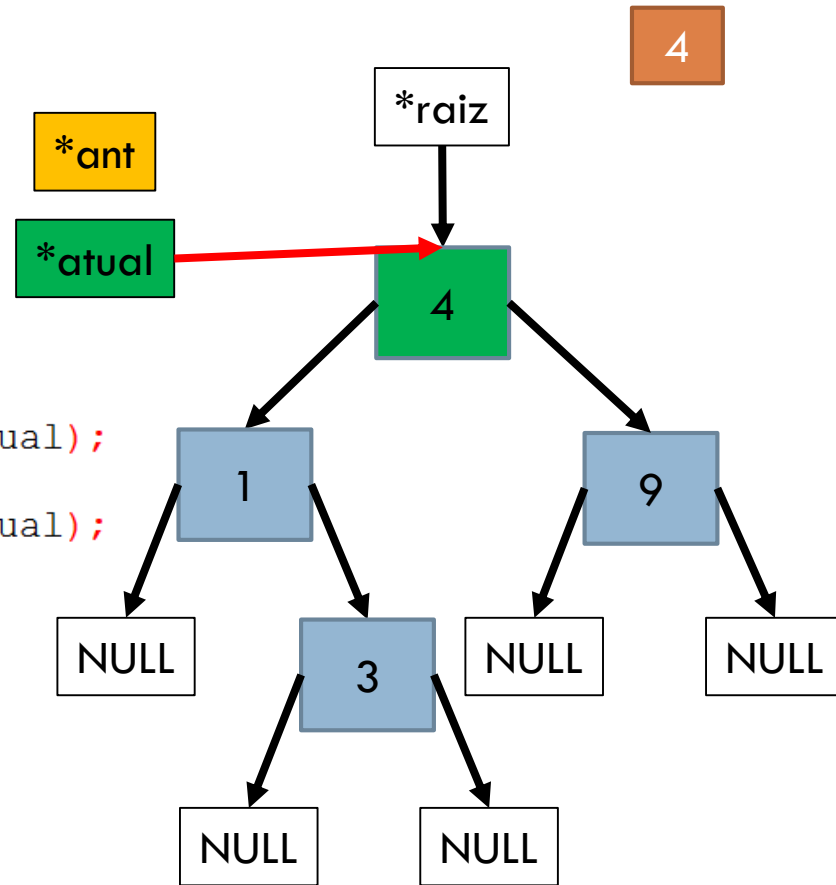
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

199

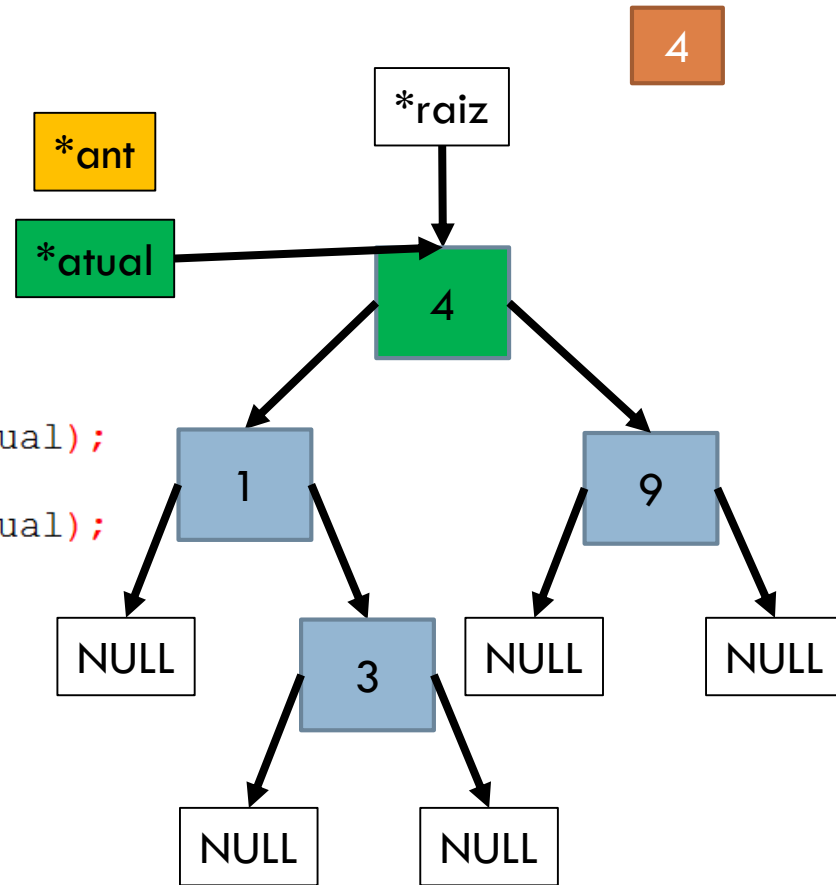
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    → struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

200

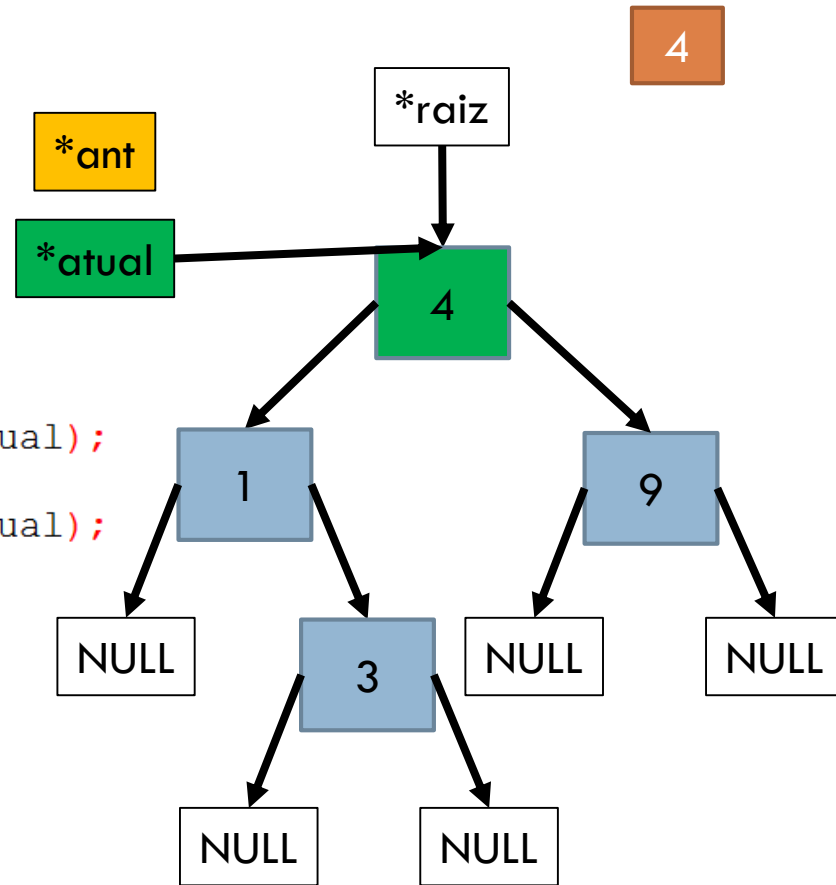
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    if(raiz == NULL)  
        return 0;  
    struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    → while(atual != NULL) {  
        if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

201

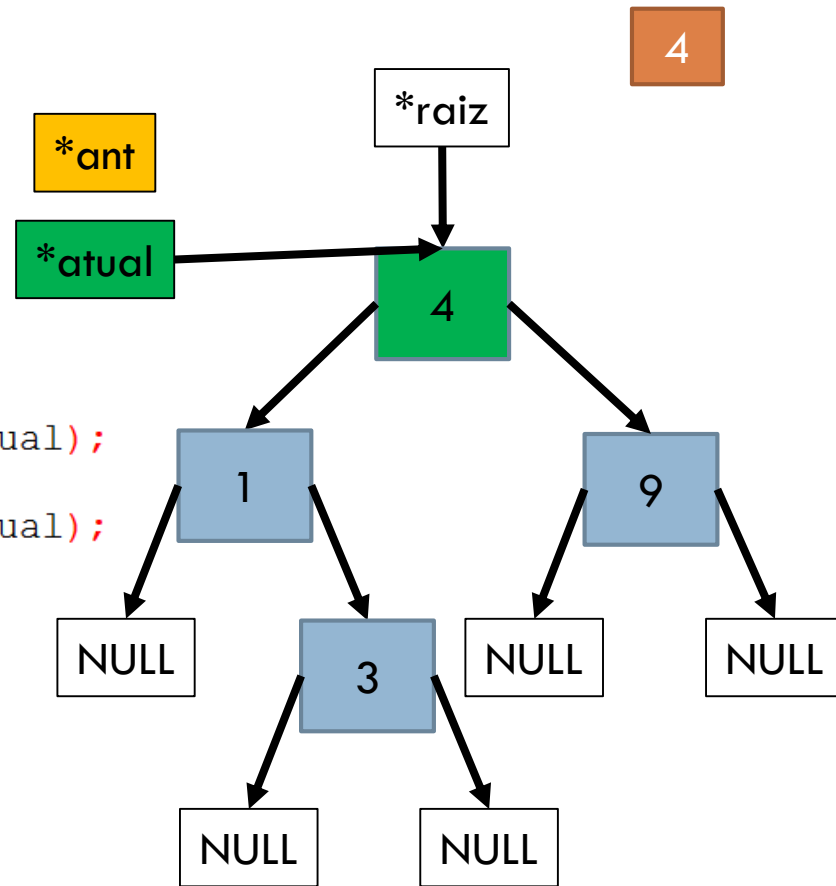
```
int remove_ArvBin(ArvBin *raiz, int valor) {  
    if(raiz == NULL)  
        return 0;  
    struct NO* ant = NULL;  
    struct NO* atual = *raiz;  
    while(atual != NULL) {  
        → if(valor == atual->info) {  
            if(atual == *raiz)  
                *raiz = remove_atual(atual);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

202

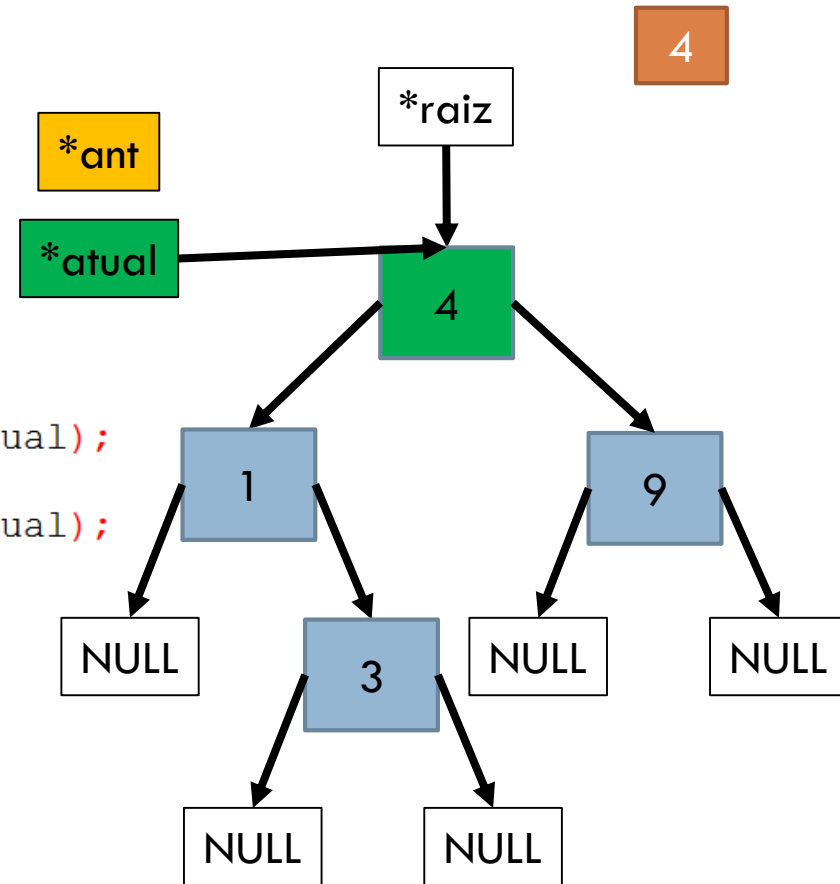
```
int remove_ArvBin(ArvBin *raiz, int valor) {
    if(raiz == NULL)
        return 0;
    struct NO* ant = NULL;
    struct NO* atual = *raiz;
    while(atual != NULL) {
        if(valor == atual->info) {
            → if(atual == *raiz)
                *raiz = remove_atual(atual);
            else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

203

```
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;
}
```

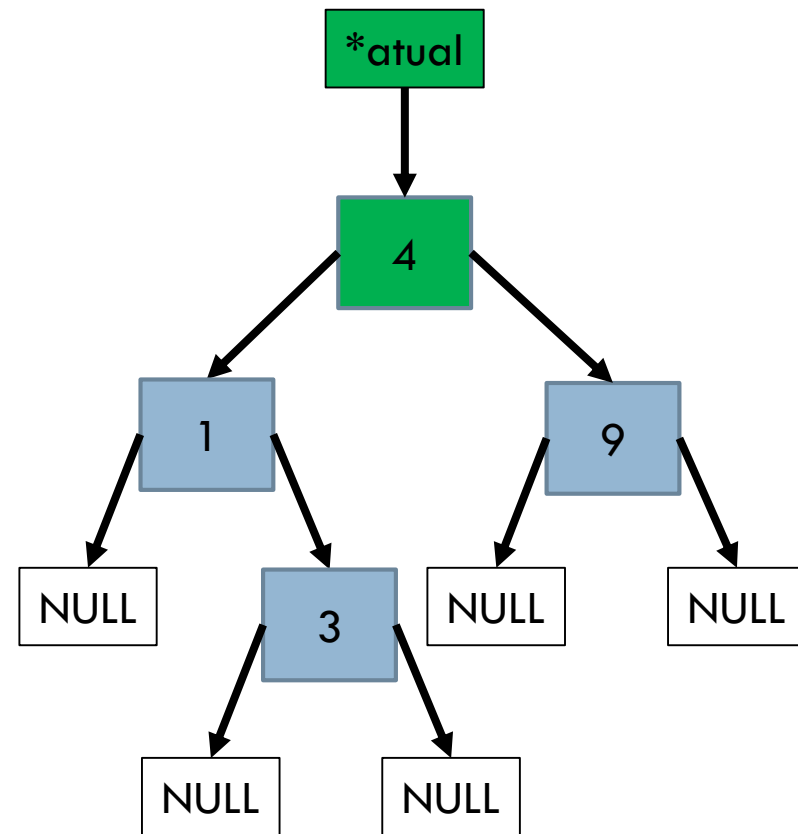


Remoção em ABB

204

```
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;
    }

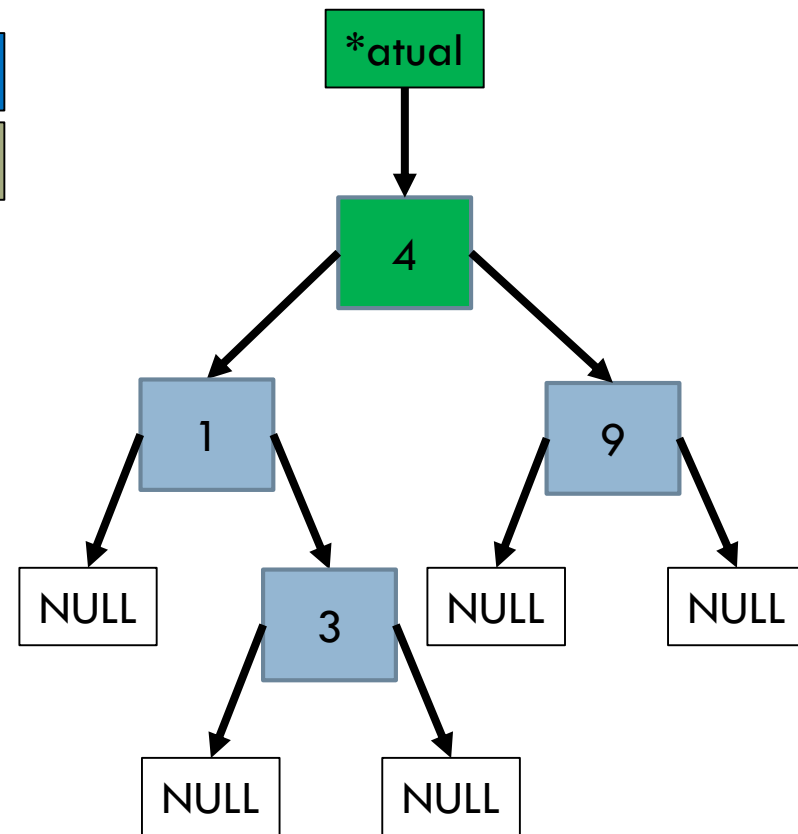
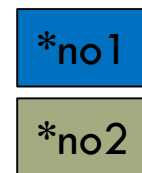
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

205

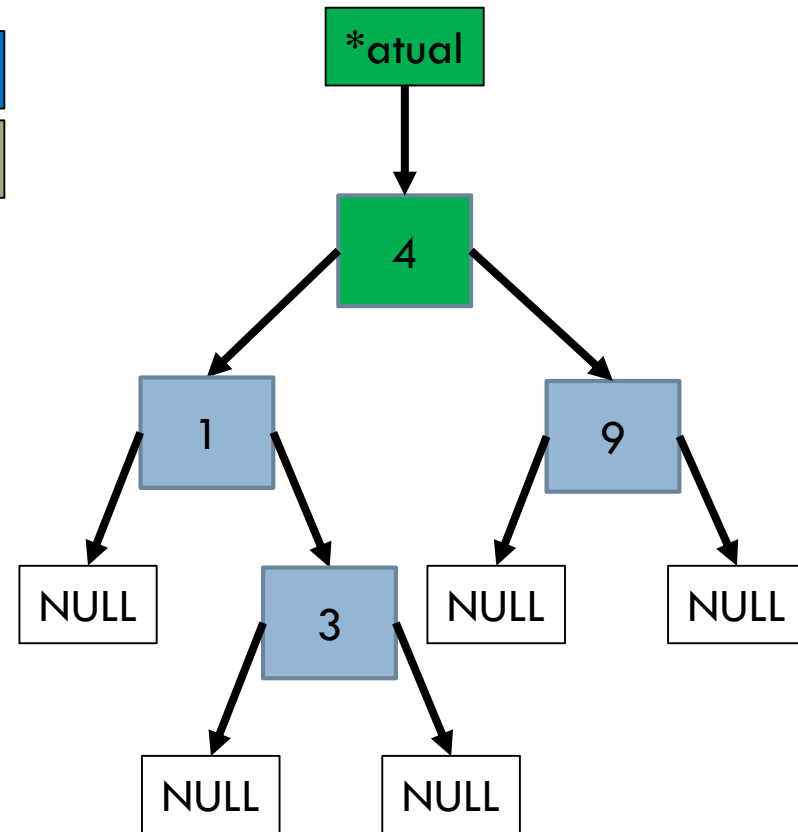
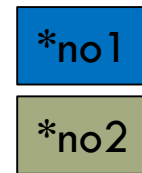
```
struct NO* remove_atual(struct NO* atual) {  
    → struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 =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



Remoção em ABB

206

```
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;  
}
```

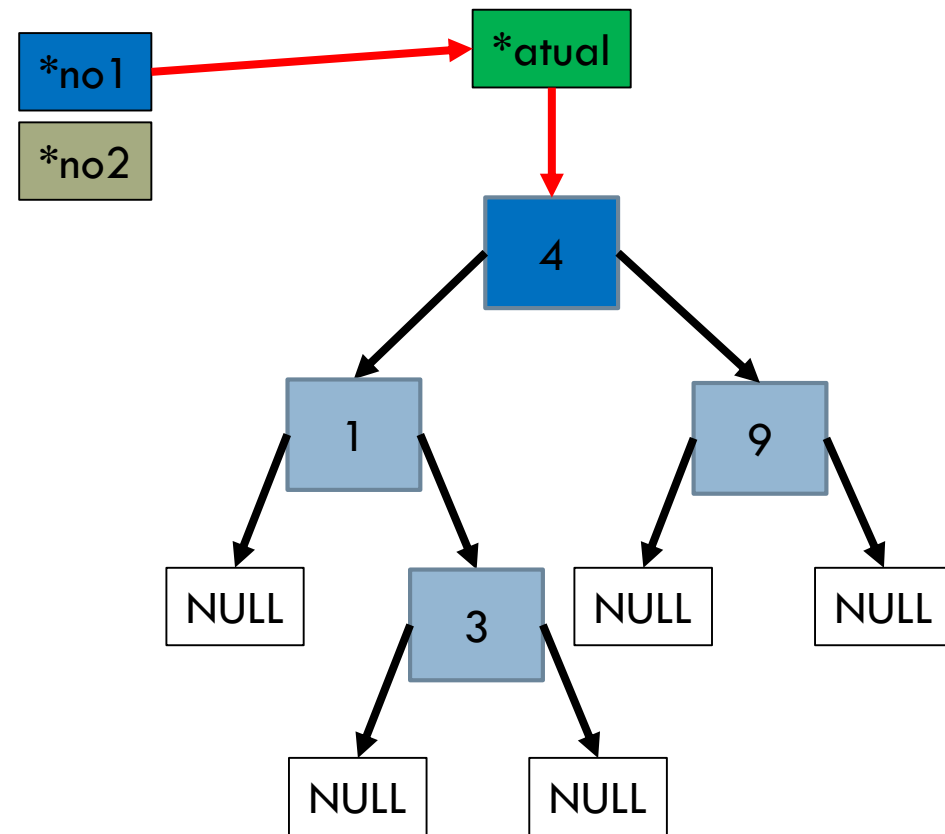


Remoção em ABB

207

```
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;  
}
```

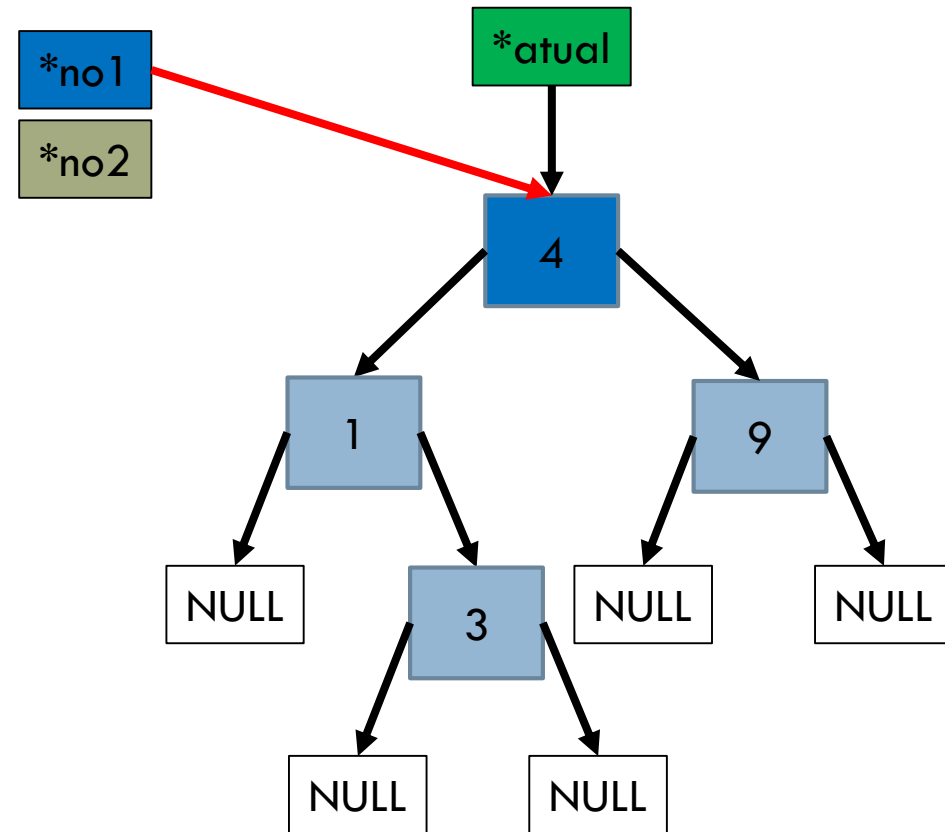


Remoção em ABB

208

```
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;  
}
```

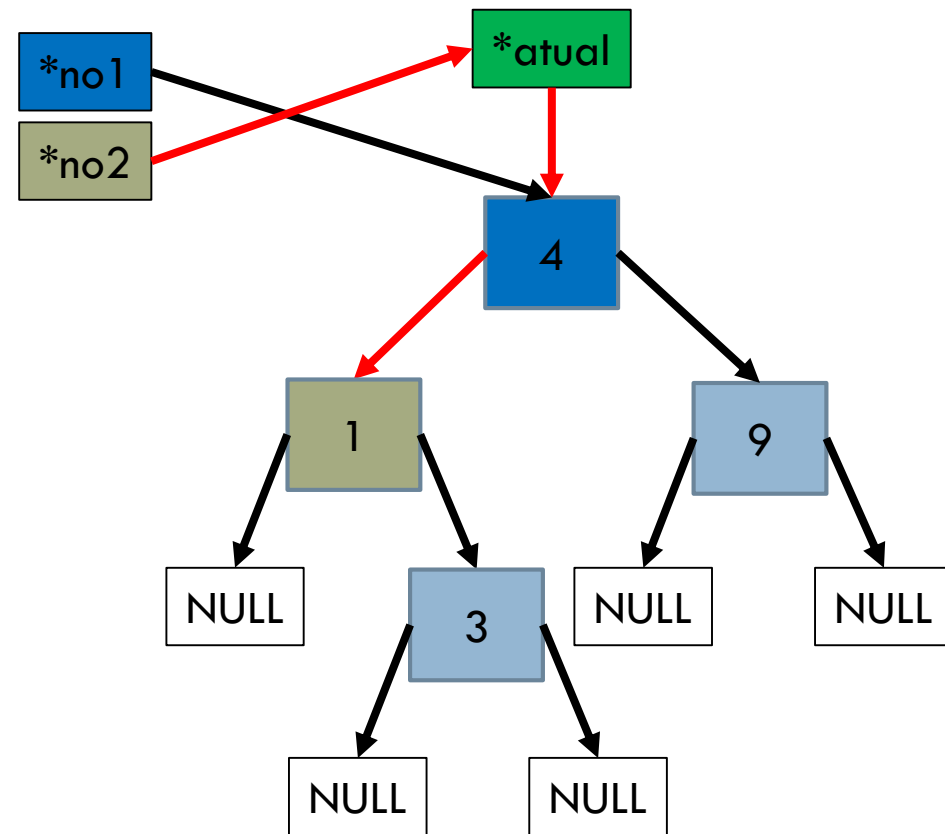


Remoção em ABB

209

```
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;
}
```

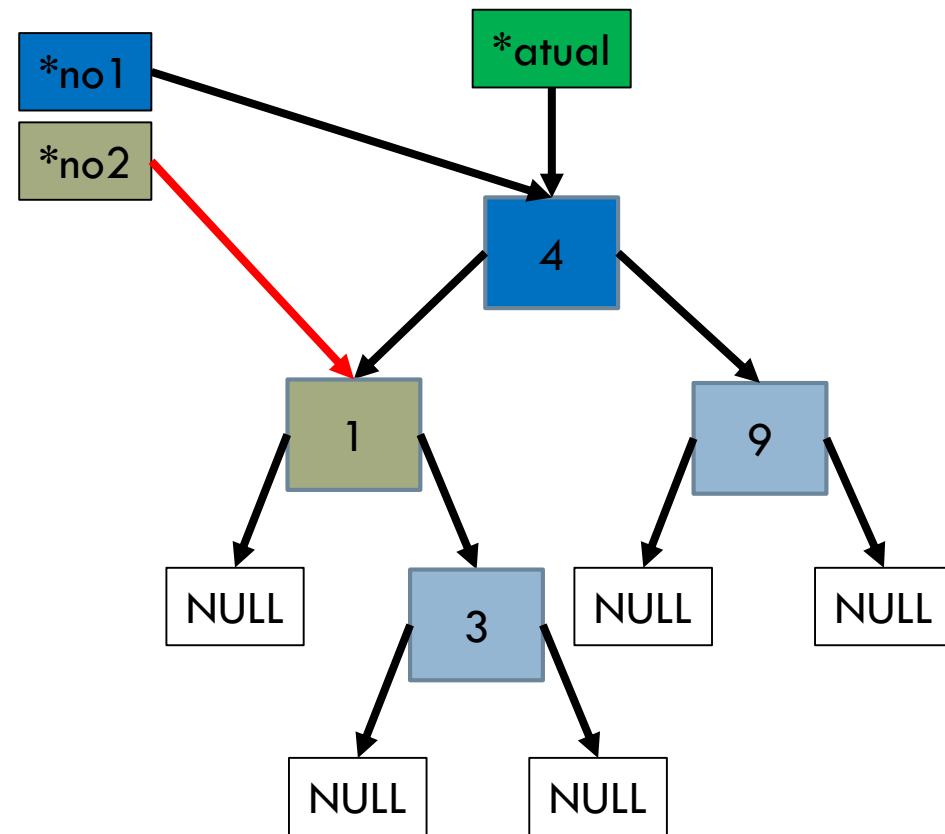


Remoção em ABB

210

```
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;
}
```

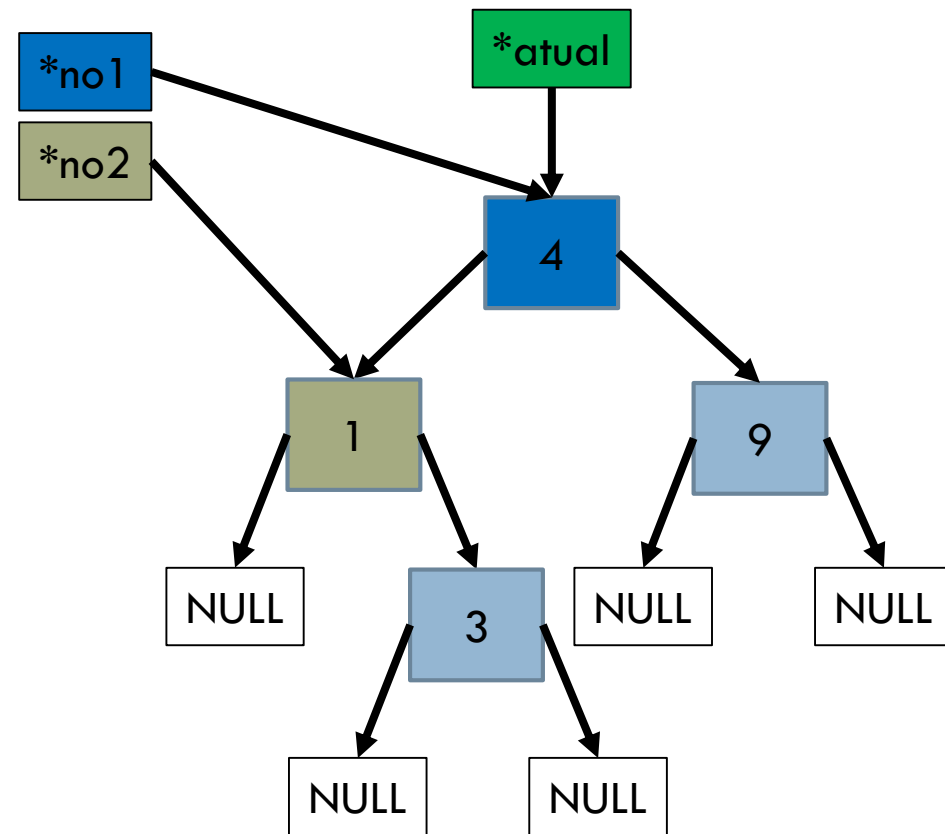


Remoção em ABB

211

```
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;
    }

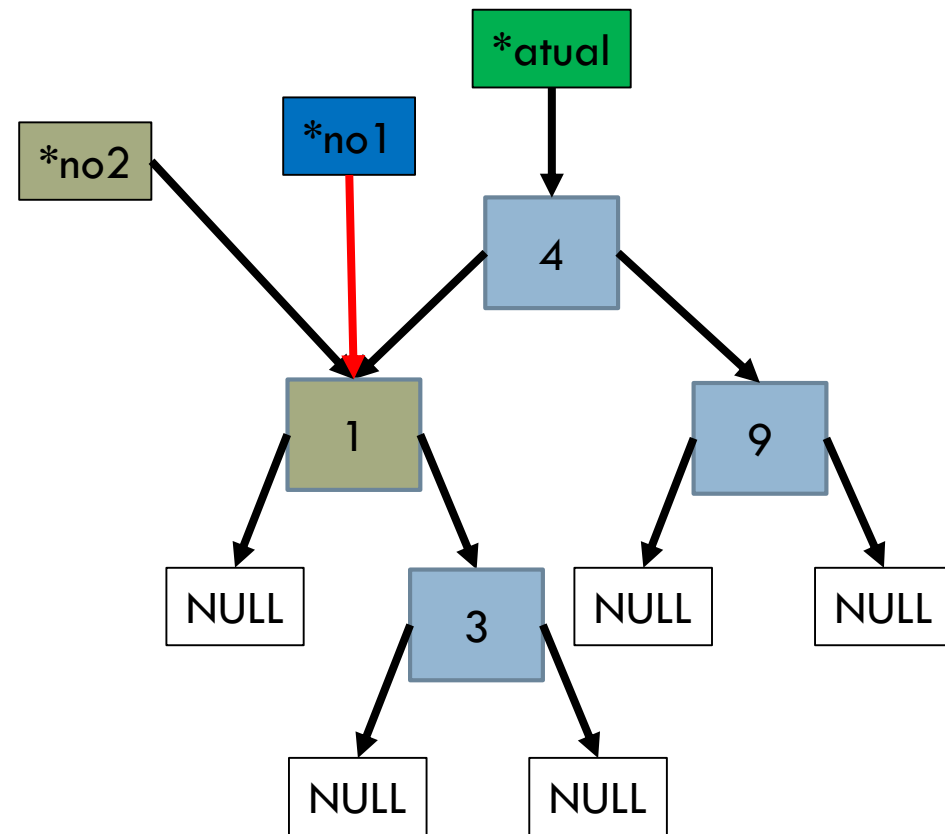
    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

212

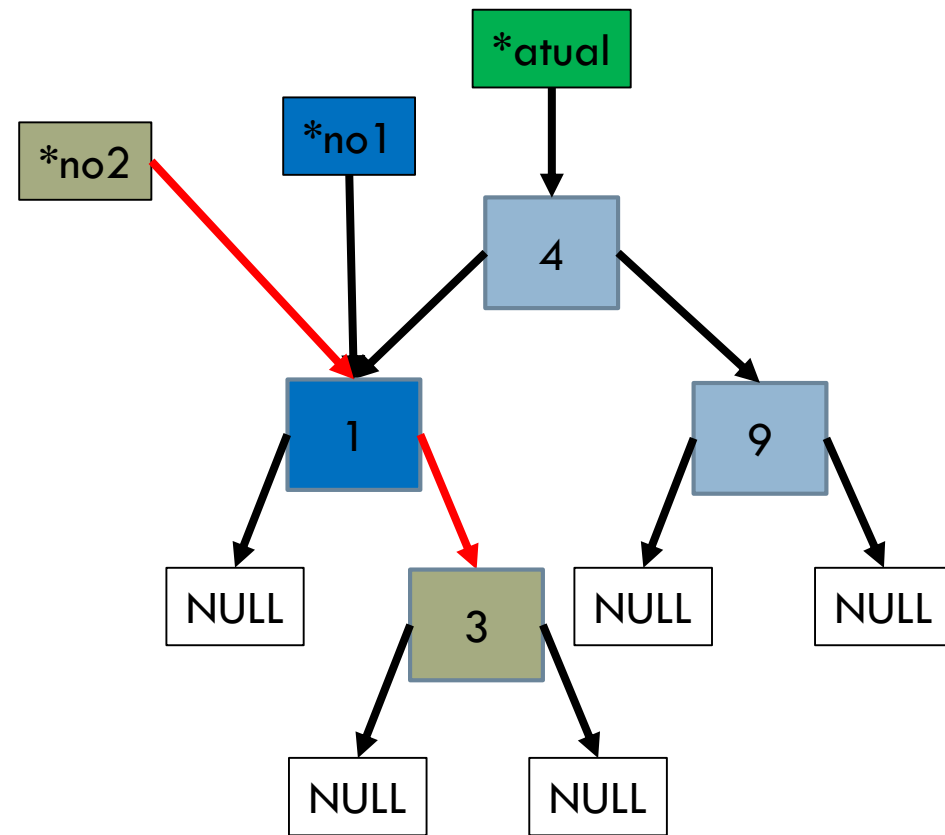
```
struct NO* remove_atual(struct NO* atual) {  
    struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        → no1 = no2;  
        no2 =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



Remoção em ABB

213

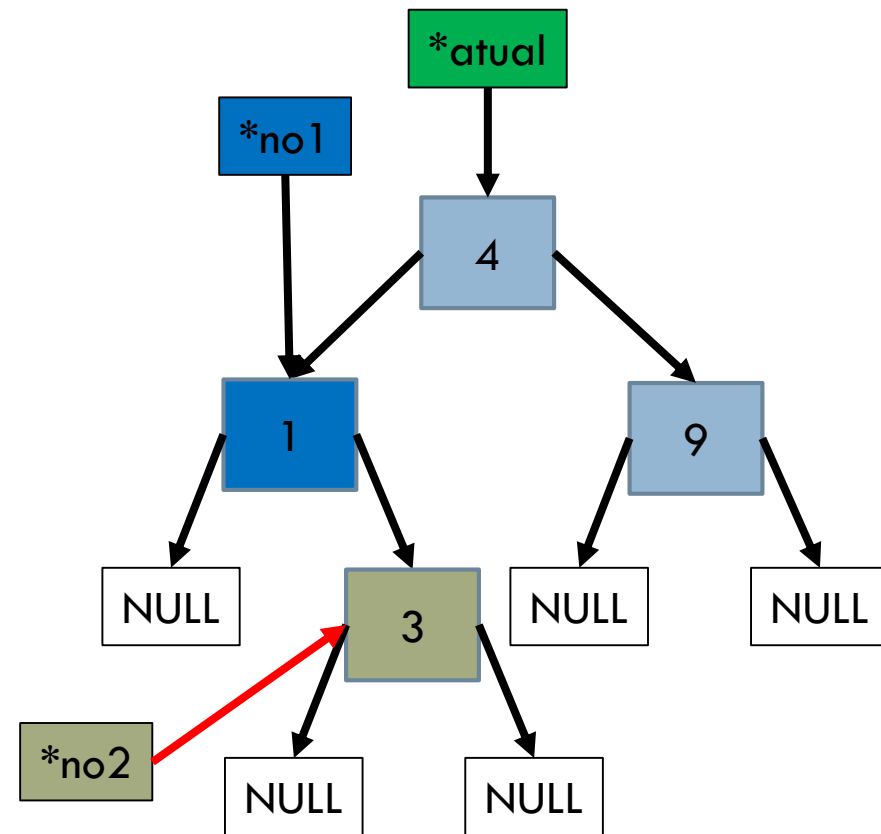
```
struct NO* remove_atual(struct NO* atual) {  
    struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        → no2 =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



Remoção em ABB

214

```
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;  
}
```

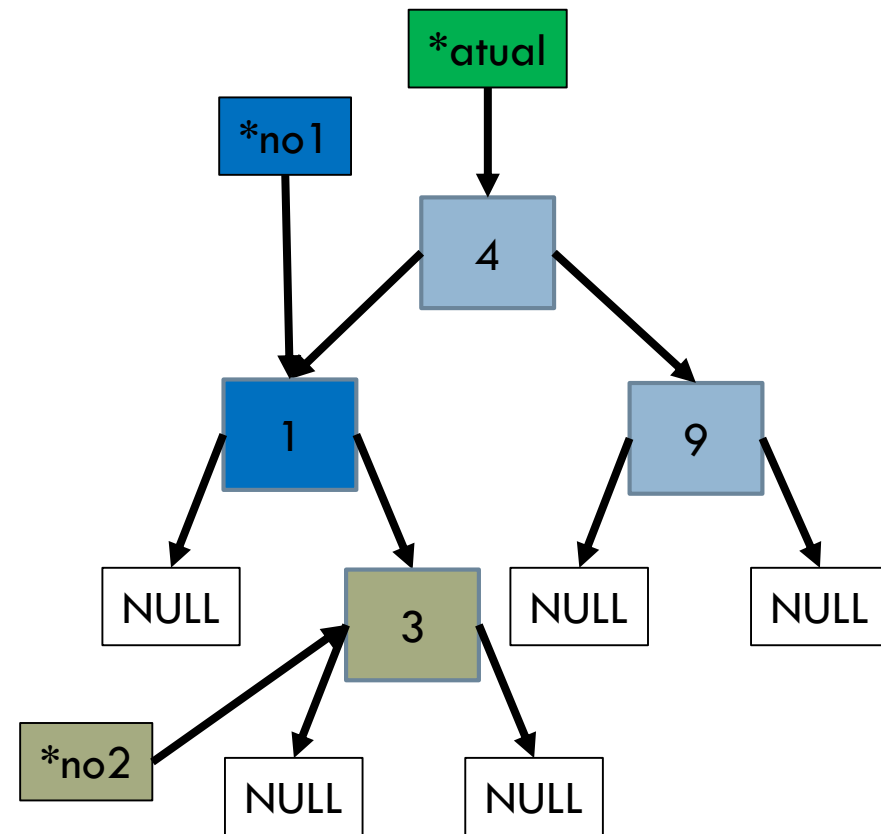


Remoção em ABB

215

```
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;
    }

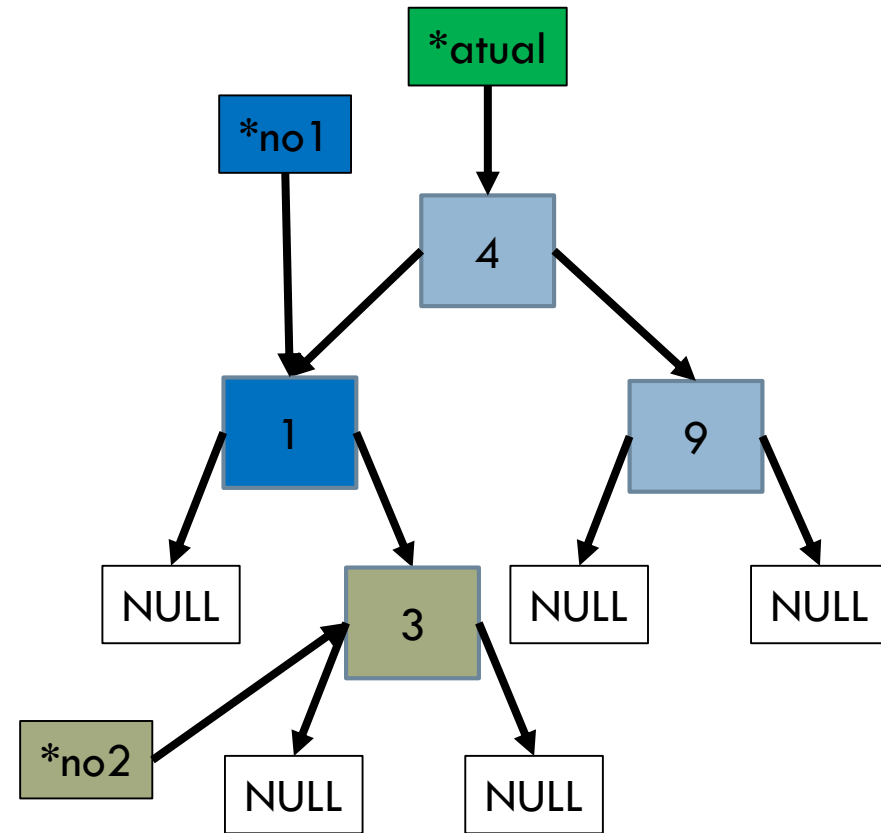
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    free(atual);
    return no2;
}
```



Remoção em ABB

216

```
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;
}
```

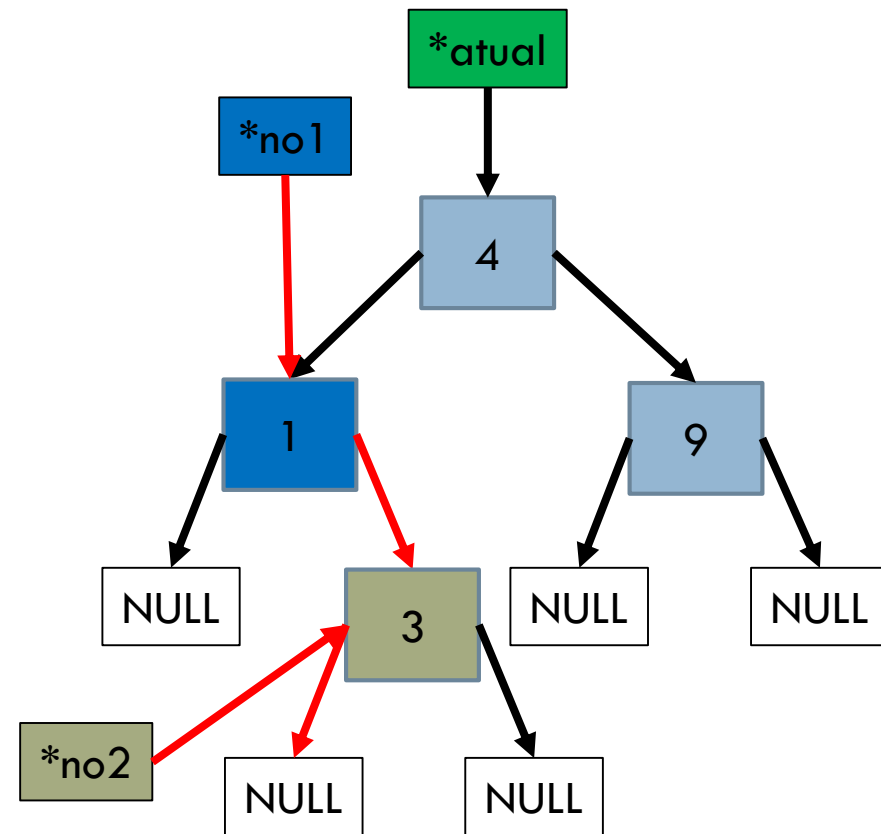


Remoção em ABB

217

```
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;
}
```

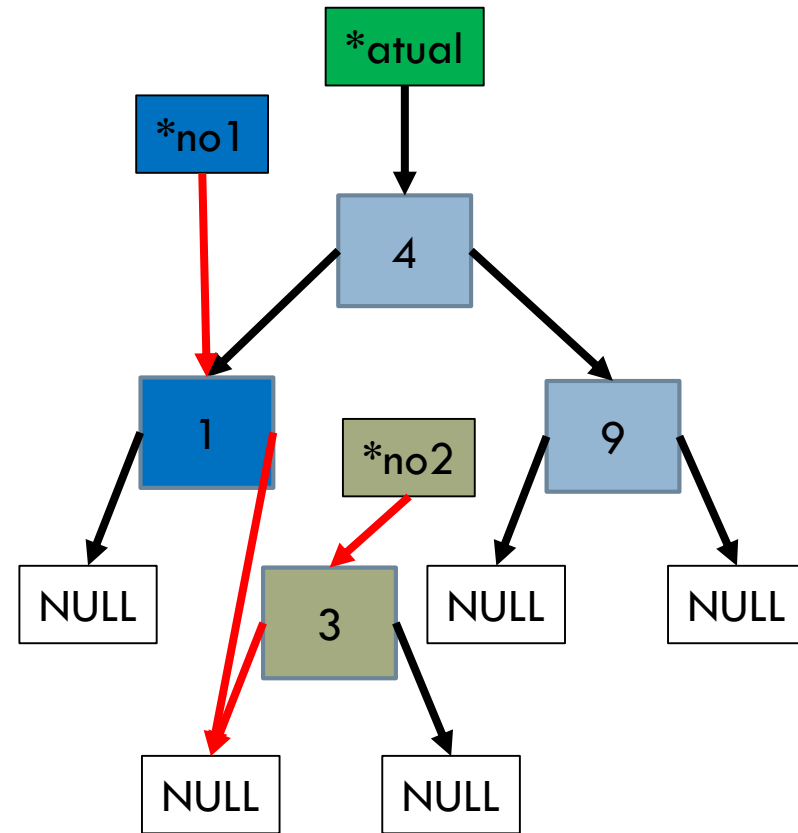


Remoção em ABB

218

```
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;
}
```

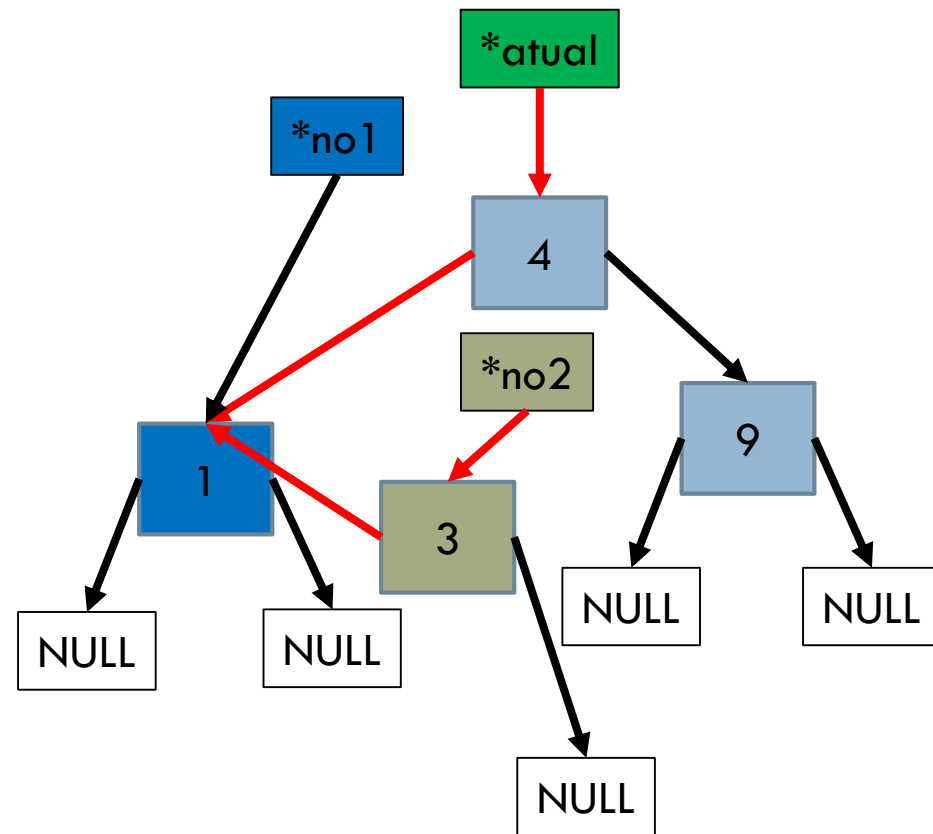


Remoção em ABB

219

```
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;
}
```

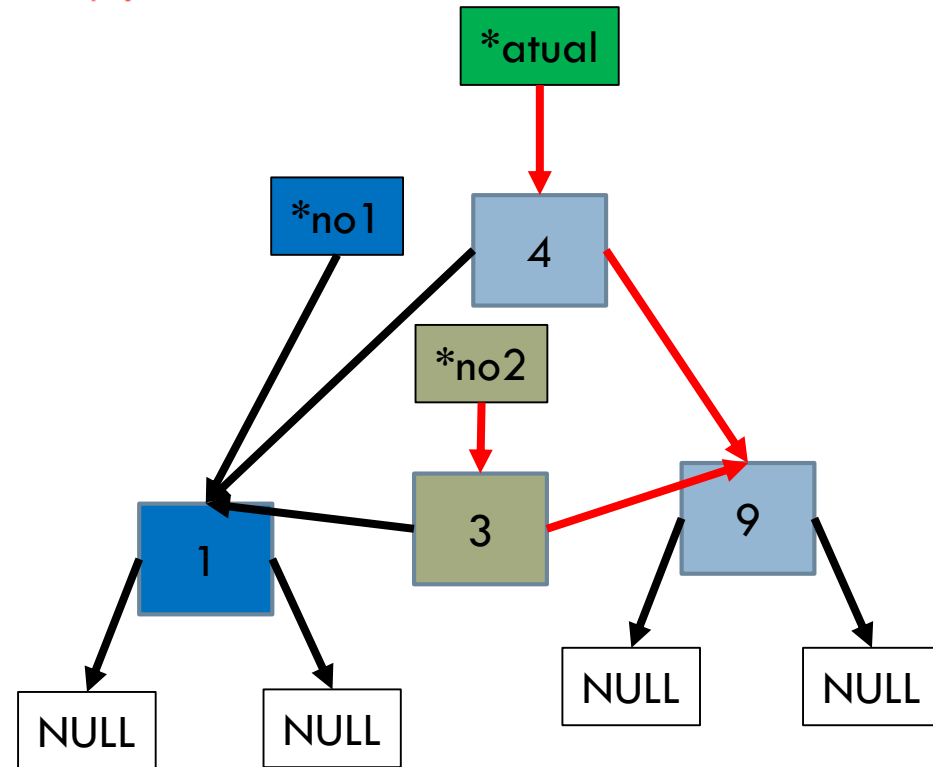


Remoção em ABB

220

```
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;
}
```

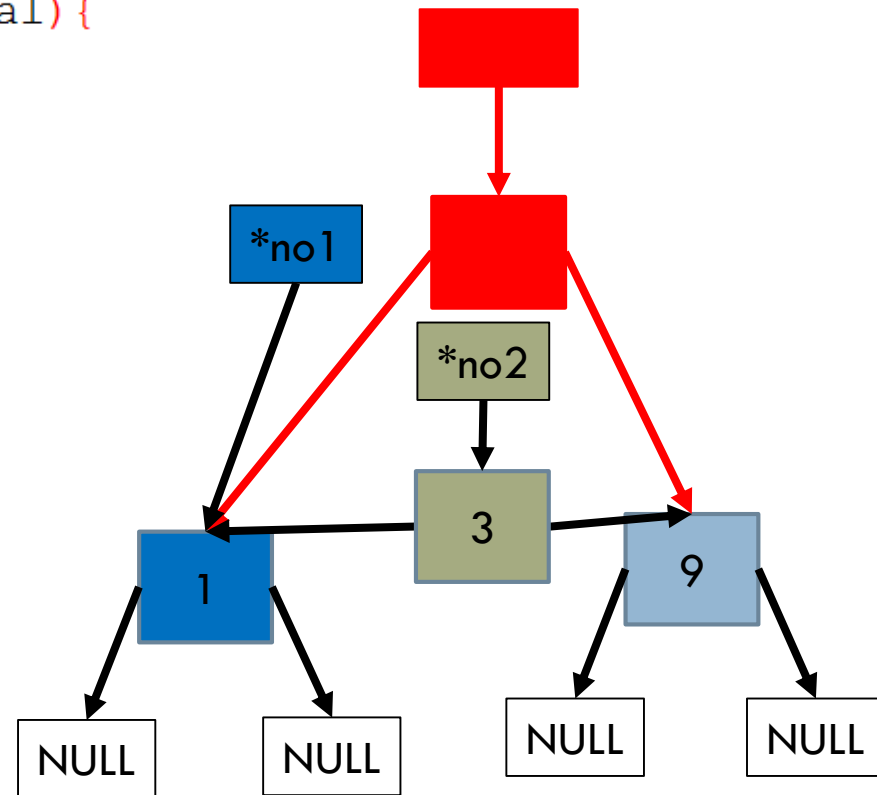


Remoção em ABB

221

```
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;
}
```

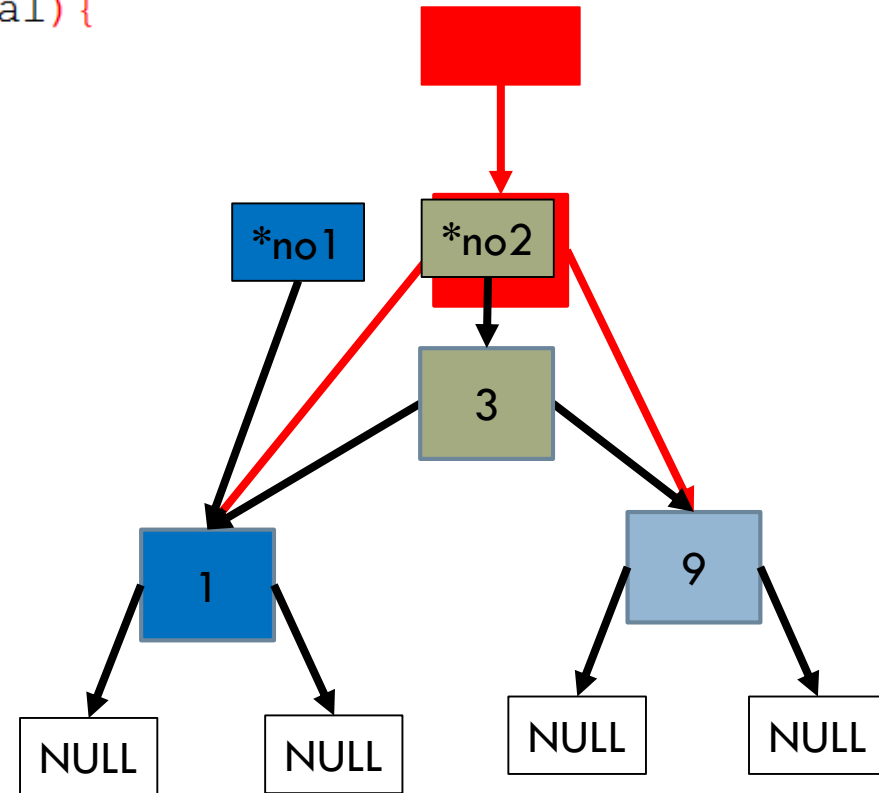


Remoção em ABB

222

```
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;
}
```

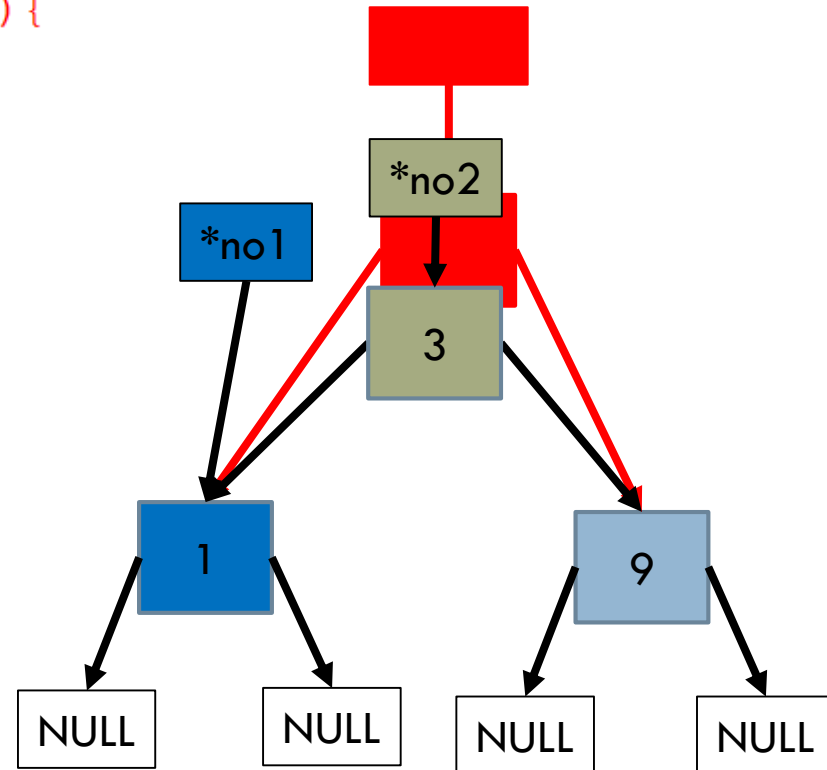


Remoção em ABB

223

```
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;
    }

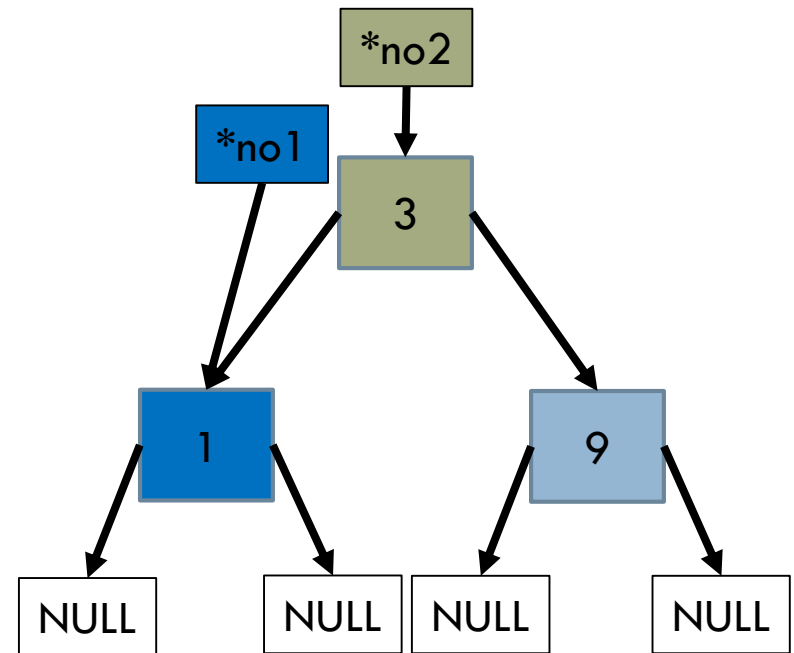
    if(no1 != atual) {
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    → free(atual);
    return no2;
}
```



Remoção em ABB

224

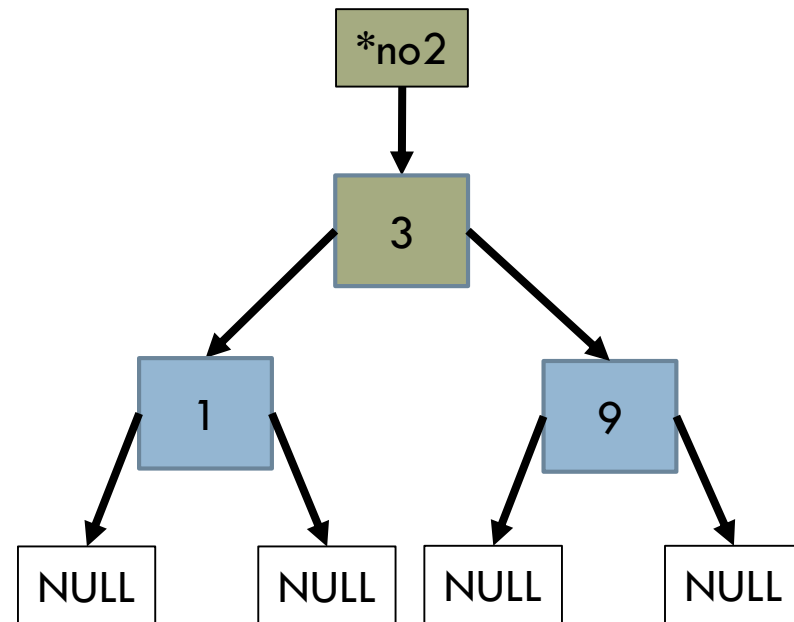
```
struct NO* remove_atual(struct NO* atual) {  
    struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 = no2->dir;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



Remoção em ABB

225

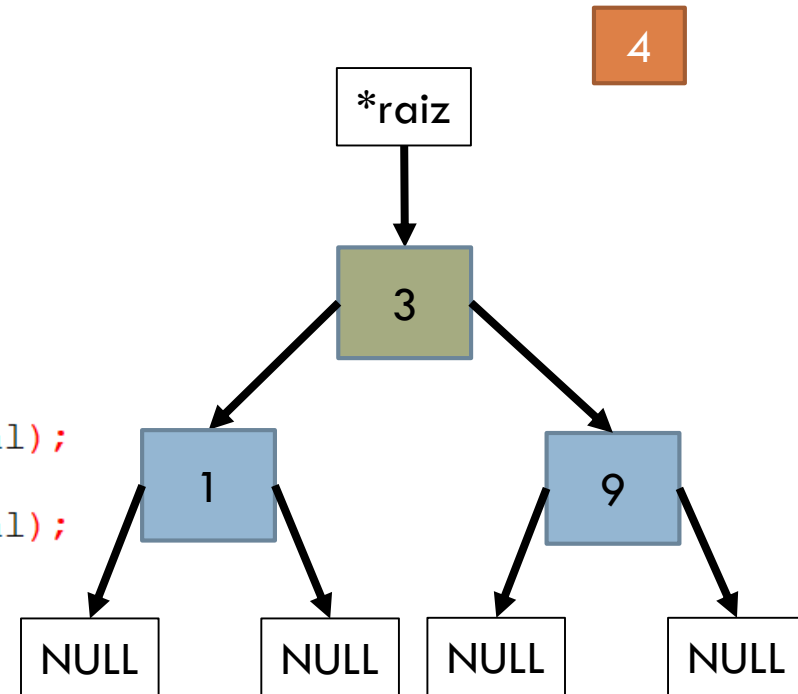
```
struct NO* remove_atual(struct NO* atual) {  
    struct NO *no1, *no2;  
    if(atual->esq == NULL) {  
        no2 = atual->dir;  
        free(atual);  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq;  
    while(no2->dir != NULL) {  
        no1 = no2;  
        no2 = no2->dir;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



Remoção em ABB

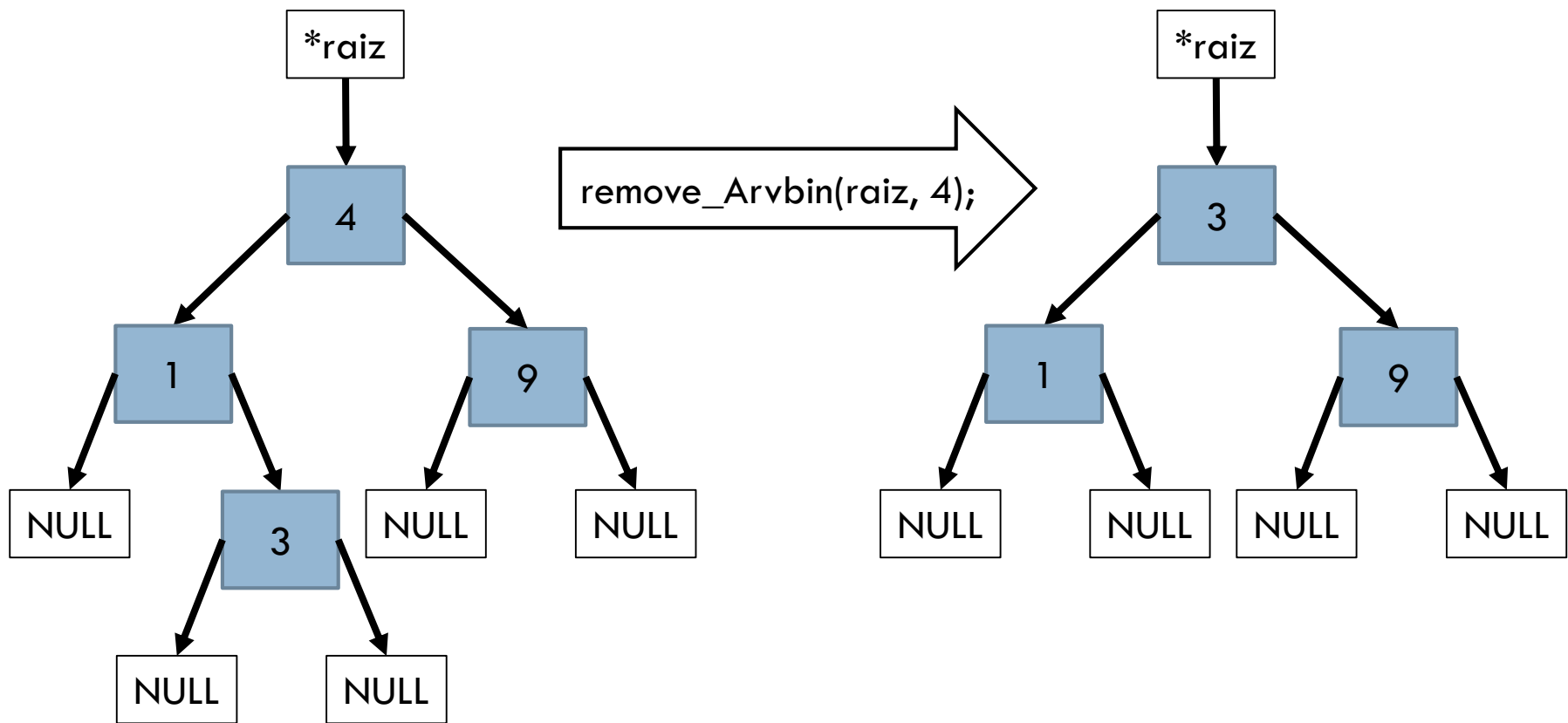
226

```
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

227



Exercícios

228

- 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

229

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

Balanceamento de ABB

230

- 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

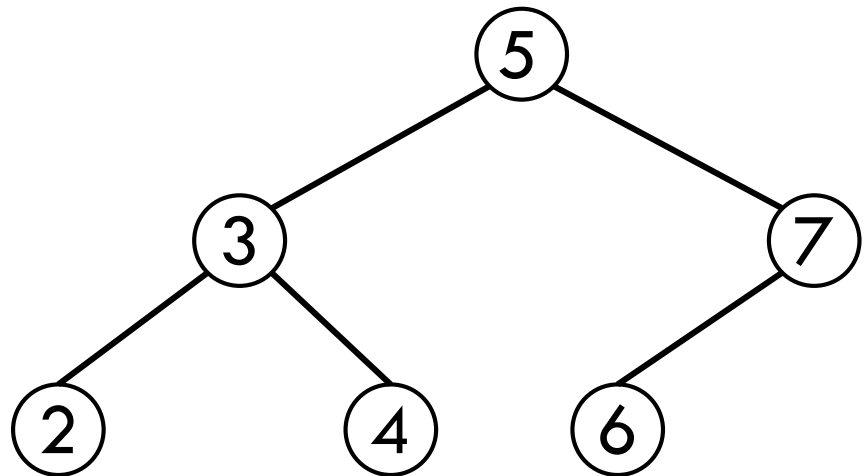
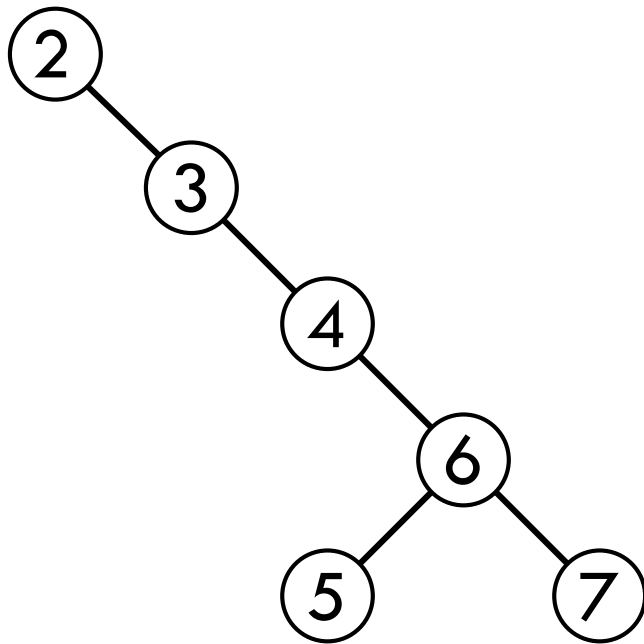
231

- 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

232

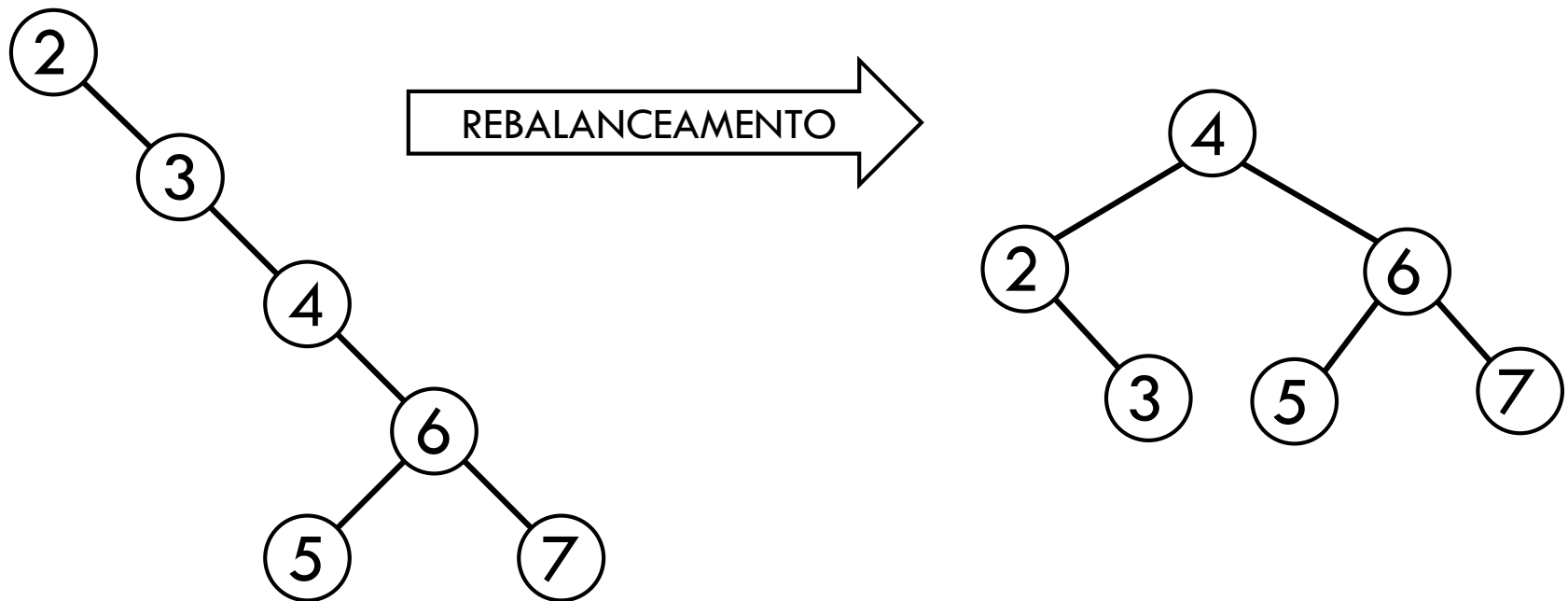
□ Exemplos de árvores: 2-3-4-5-6-7



Rebalanceamento de ABB

233

- Reorganizar uma ABB para ficar balanceada.



Rebalanceamento de ABB

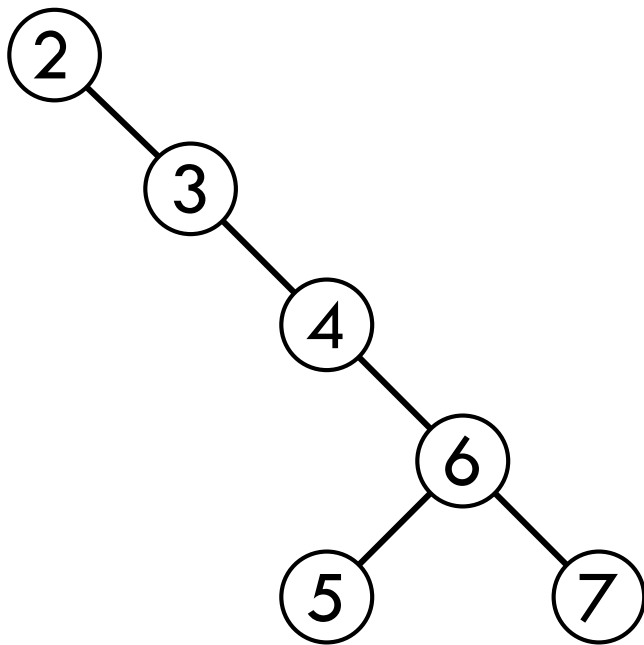
234

- 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

235

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

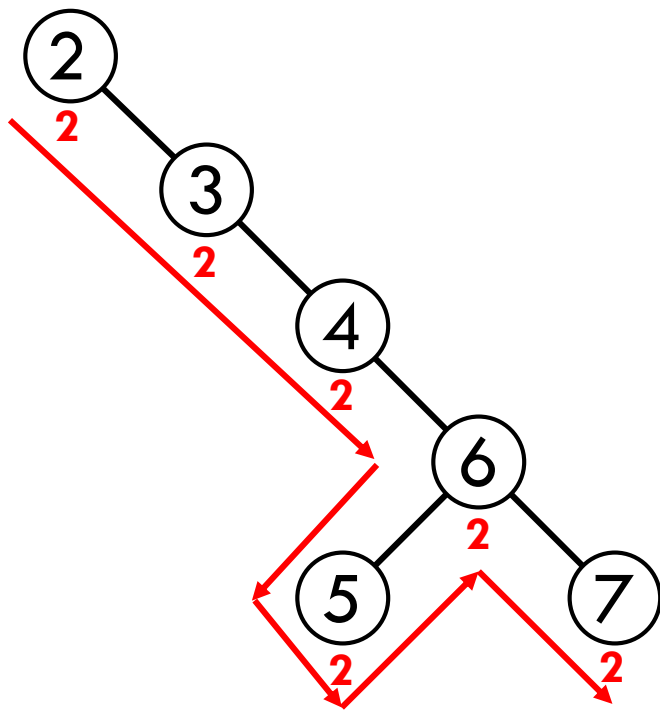


0	1	2	3	4	5

Rebalanceamento de ABB

236

- 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

237

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

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

	ÍNDICE
INÍCIO	
FIM	
MEIO	

Rebalanceamento de ABB

238

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

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

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	

Rebalanceamento de ABB

239

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

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

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	2

Rebalanceamento de ABB

240

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

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

④

	ÍNDICE
INÍCIO	0
FIM	5
MEIO	2

Rebalanceamento de ABB

241

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

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

④

	ÍNDICE
INÍCIO	0
FIM	1
MEIO	

Rebalanceamento de ABB

242

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

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

④

	ÍNDICE
INÍCIO	0
FIM	1
MEIO	0

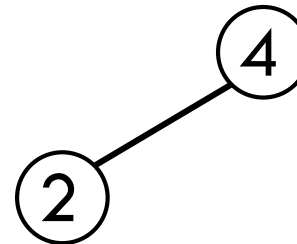
Rebalanceamento de ABB

243

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

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

	ÍNDICE
INÍCIO	0
FIM	1
MEIO	0



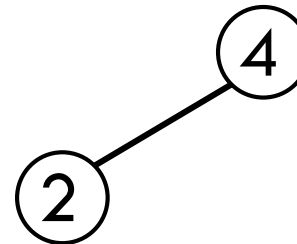
Rebalanceamento de ABB

244

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

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

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	



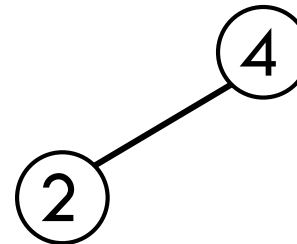
Rebalanceamento de ABB

245

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

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

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	1



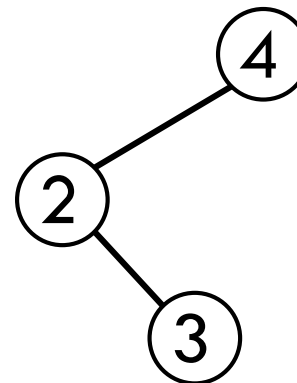
Rebalanceamento de ABB

246

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

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

	ÍNDICE
INÍCIO	1
FIM	1
MEIO	1



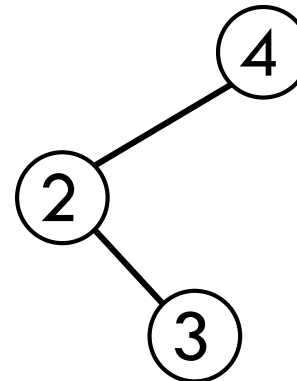
Rebalanceamento de ABB

247

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

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

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	



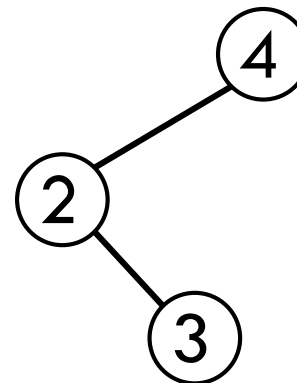
Rebalanceamento de ABB

248

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

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

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	4



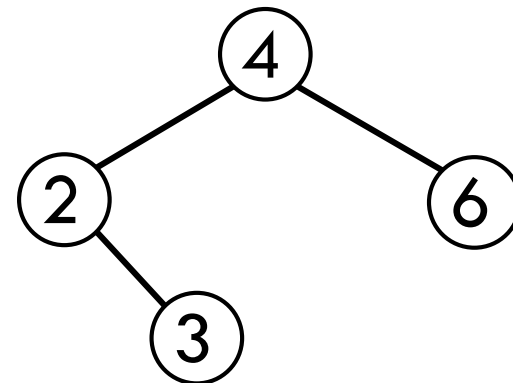
Rebalanceamento de ABB

249

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

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

	ÍNDICE
INÍCIO	3
FIM	5
MEIO	4



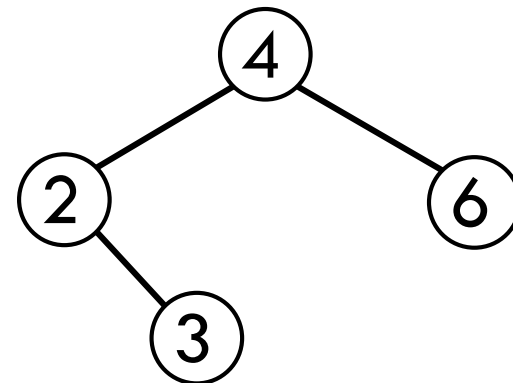
Rebalanceamento de ABB

250

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

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

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	



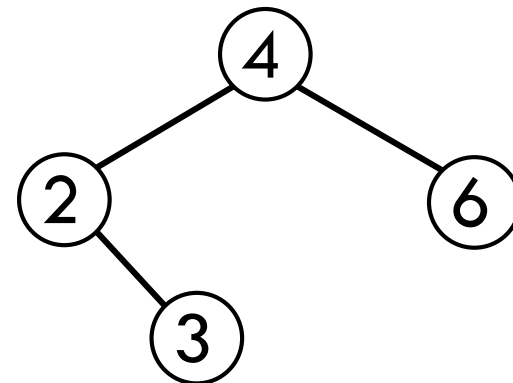
Rebalanceamento de ABB

251

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

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

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	3



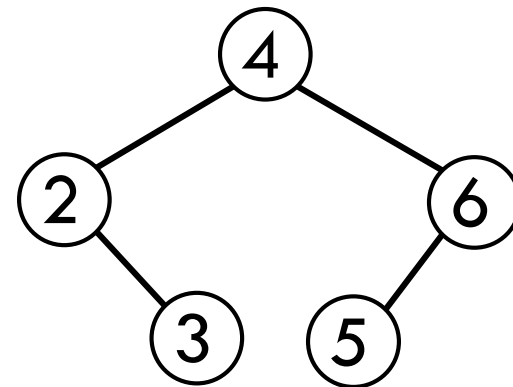
Rebalanceamento de ABB

252

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

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

	ÍNDICE
INÍCIO	3
FIM	3
MEIO	3



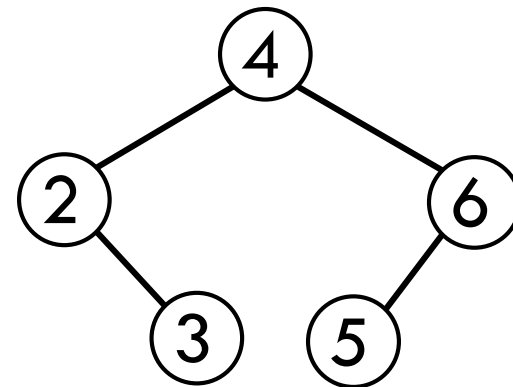
Rebalanceamento de ABB

253

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

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

	ÍNDICE
INÍCIO	5
FIM	5
MEIO	



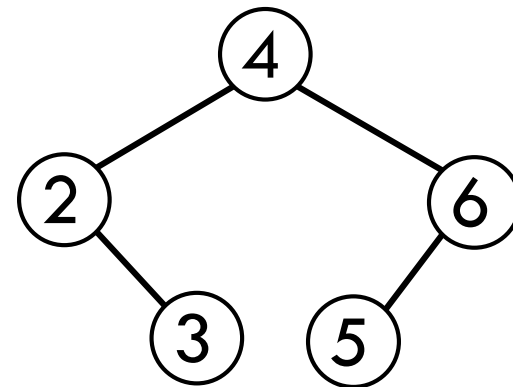
Rebalanceamento de ABB

254

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

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

	ÍNDICE
INÍCIO	5
FIM	5
MEIO	5



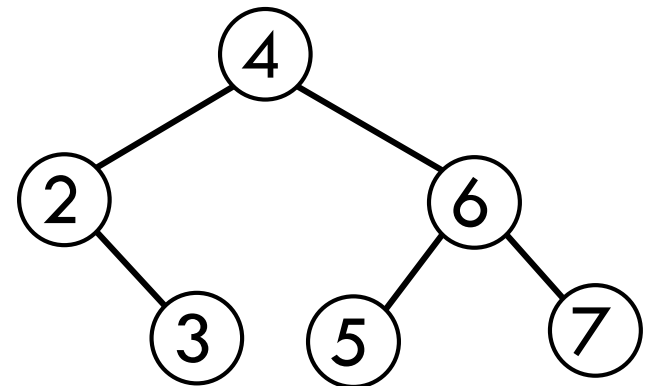
Rebalanceamento de ABB

255

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

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

	ÍNDICE
INÍCIO	5
FIM	5
MEIO	5



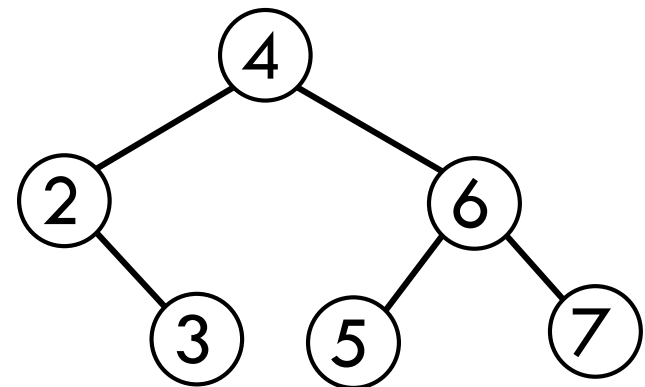
Rebalanceamento de ABB

256

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

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

	ÍNDICE
INÍCIO	-
FIM	-
MEIO	-



Exercício

257

- 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.