# Árvores – Árvore binária de busca

Prof. Renato Pimentel

2025/2



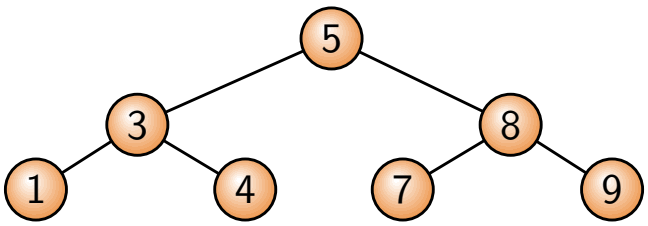
## Sumário

1. Árvore binária de busca
   - Definição
   - Inserção na ABB
   - Busca na ABB
   - Remoção na ABB
   - Exercícios

# Árvore binária de busca

- **Árvore binária** – cada nó possui 0, 1 ou 2 filhos.
- Comumente referenciada pela sigla **BST** – *binary search tree*.
- Todo nó possui um **valor** (**chave**) associado.
  - Não existem chaves repetidas.
- Chave determina a **posição** do nó na ABB. Para cada nó pai:
  - todos os valores da **subárvore da esquerda** são **menores** do que o nó pai
  - todos os valores da **subárvore da direita** são **maiores** do que o nó pai;
- Inserção e remoção devem ser realizadas respeitando tal regra.





- Ótima alternativa para operações de busca binária.
  - Possui a vantagem de ser uma estrutura dinâmica em comparação ao *array* (*heap*);
  - Mais fácil inserir valores na árvore do que num *array* ordenado:
    - ★ *Array*: envolve deslocamento de elementos.



## Custo computacional assintótico

- Custo para as principais operações em uma árvore binária de busca contendo $n$ nós.
  - O pior caso ocorre quando a ABB não está **balanceada**.

| | **Melhor Caso** | **Pior Caso** |
|---|---|---|
| **Inserção** | $O(\log_2 n)$ | $O(n)$ |
| **Busca** | $O(\log_2 n)$ | $O(n)$ |
| **Remoção** | $O(\log_2 n)$ | $O(n)$ |

No TAD Árvore binária:

ArvoreBinaria.h

```c
typedef struct NO* ArvBin;

ArvBin* cria_ArvBin();
void libera_ArvBin(ArvBin *raiz);
int estaVazia_ArvBin(ArvBin *raiz);
int altura_ArvBin(ArvBin *raiz);
int totalNO_ArvBin(ArvBin *raiz);
int insere_ArvBin(ArvBin* raiz, int valor);
int consulta_ArvBin(ArvBin *raiz, int valor);
int remove_ArvBin(ArvBin *raiz, int valor);
```



# Sumário

1. Árvore binária de busca
   - Definição
   - Inserção na ABB
   - Busca na ABB
   - Remoção na ABB
   - Exercícios

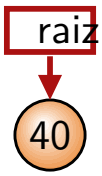
# Inserção na ABB

- Para inserir um valor $V$ na ABB:
  1. Se a raiz é igual a `NULL`, insira o nó;
  2. Se $V$ é **menor que** a raiz: siga para a **subárvore da esquerda**;
  3. Se $V$ é **maior que** a raiz: siga para a **subárvore da direita**;
  4. Aplique o método recursivamente – também é possível sem recursão.
- Percorre-se os nós da ABB até o **nó folha** que se tornará **pai** do novo nó.



- Também deve se considerar inserção numa árvore vazia;
- Neste caso, novo elemento será a **raiz** da ABB.

Insere valor 40
em uma árvore vazia




*raiz = novo;

Fonte: adaptado de BACKES (2023).

## Exemplo de uso

```
int N = 8, dados[8] = {50,100,30,20,40,45,35,37};
ArvBin* raiz = cria_ArvBin();
int i;
for(i=0; i<N; i++)
  insere_ArvBin(raiz, dados[i]);
```

## ArvoreBinaria.c: inserção de nó

```
65 int insere_ArvBin(ArvBin* raiz, int valor){
66   if(raiz == NULL)
67     return 0;
68   struct NO* novo;
69   novo = (struct NO*) malloc(sizeof(struct NO));
70   if(novo == NULL)
71     return 0;
72   novo->info = valor;
73   novo->dir = NULL;
74   novo->esq = NULL;
```



```
75
76   if(*raiz == NULL)
77     *raiz = novo;
78   else{
79     struct NO* atual = *raiz;
80     struct NO* ant = NULL;
81
82     // Navega nos nós da árvore até chegar em um nó
     folha
83     while(atual != NULL){
84       ant = atual;
85       if(valor == atual->info){
86         free(novo);
87         return 0; // elemento já existe
88       }
89
90       if(valor > atual->info)
91         atual = atual->dir;
92       else
93         atual = atual->esq;
```

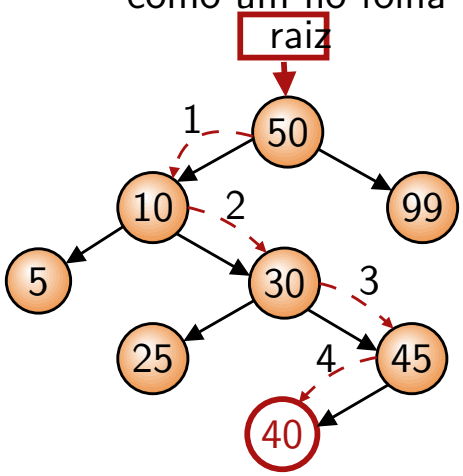
```
    }

    // Insere como filho desse nó folha
    if(valor > ant->info)
      ant->dir = novo;
    else
      ant->esq = novo;
  }
  return 1;
}
```



## Passo-a-passo

Insere o valor 40
como um nó folha




| | |
|---|---|
| 1 | valor é menor do que 50: visita filho da esquerda |
| 2 | valor é maior do que 10: visita filho da direita |
| 3 | valor é maior do que 30: visita filho da direita |
| 4 | valor é menor do que 45: visita filho da esquerda |
| fim | Não existe filho da esquerda. Valor passa a ser o filho da esquerda de 45 |

Fonte: BACKES (2023).

# Sumário


1. Árvore binária de busca
   - Definição
   - Inserção na ABB
   - Busca na ABB
   - Remoção na ABB
   - Exercícios




# Busca na ABB

- Consultar se um determinado nó $V$ existe em uma ABB é similar a operação de inserção:
  1. Inicialmente, compare o valor buscado com a **raiz**;
  2. Se $V$ é **menor que** a raiz: siga para a **subárvore da esquerda**;
  3. Se $V$ é **maior que** a raiz: siga para a **subárvore da direita**;
  4. Aplique o método recursivamente até que a raiz seja igual ao valor buscado, ou `NULL`– também é possível sem recursão.

## Exemplo de uso

```
int N = 8, dados[8] = {50,100,30,20,40,45,35,37};
ArvBin* raiz = cria_ArvBin();
int i;
for(i=0; i<N; i++)
  insere_ArvBin(raiz, dados[i]);
.
.
if (consulta_ArvBin(raiz, 20))
  printf("20 está na árvore\n");
```

## ArvoreBinaria.c: busca por nó

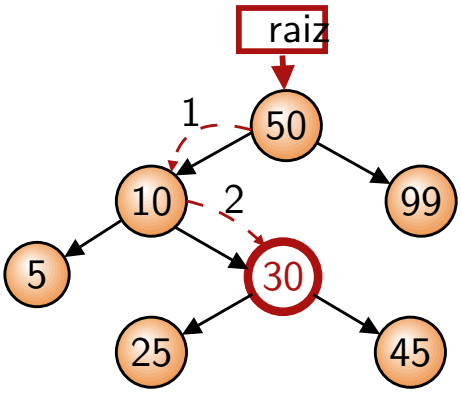
```
105 int consulta_ArvBin(ArvBin *raiz, int valor){
106   if (raiz == NULL)
107     return 0;
108   struct NO* atual = *raiz;
109   while (atual != NULL){
110     if (valor == atual->info){
```



```
111       return 1;
112     }
113     if (valor > atual->info)
114       atual = atual->dir;
115     else
116       atual = atual->esq;
117   }
118   return 0;
119 }
```

# Passo-a-passo

Valor procurado: 30

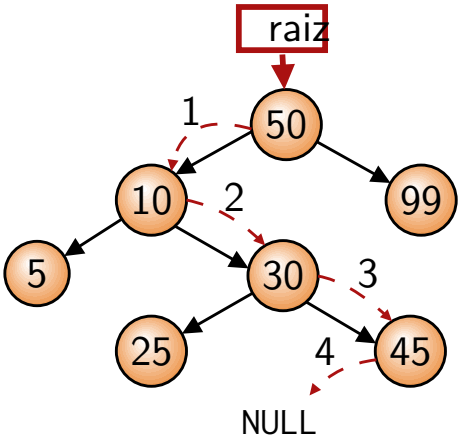



| | |
|---|---|
| 1 | valor é menor do que 50: visita filho da esquerda |
| 2 | valor é maior do que 10: visita filho da direita |
| fim | valor procurado é igual ao do nó: retornar dados do nó (ou 1) |

Fonte: BACKES (2023).



Valor procurado: 38




| | |
|---|---|
| 1 | valor é menor do que 50: visita filho da esquerda |
| 2 | valor é maior do que 10: visita filho da direita |
| 3 | valor é maior do que 30: visita filho da direita |
| 4 | valor é menor do que 45: visita filho da esquerda |
| fim | filho da esquerda de 45 não existe: busca falhou |

Fonte: BACKES (2023).

# Sumário

1 Árvore binária de busca
- Definição
- Inserção na ABB
- Busca na ABB
- Remoção na ABB
- Exercícios



# Remoção na ABB

- Remover um nó de uma árvore binária de busca não é uma tarefa tão simples quanto a inserção.
- Isso ocorre porque precisamos procurar o nó a ser removido da ABB, o qual pode ser um
  - Nó folha;
  - **Nó interno** – inclusive a raiz – com um ou dois filhos.
- Se for um nó interno:
  - Reorganizar a árvore para que ela continue sendo uma ABB.

Uso de duas funções:

- `remove_ArvBin()` – faz a busca pelo nó – disponível na interface com o usuário;
- `remove_atual()`, auxiliar, que trata os 3 tipos de remoção – com 0, 1 ou 2 filhos – reorganizando a ABB.
  - Recebe o endereço atual do nó a ser removido, e
  - retorna qual deverá ser seu substituto na ABB.

Exemplo de uso

```
int N = 8, dados[8] = {50,100,30,20,40,45,35,37};
ArvBin* raiz = cria_ArvBin();
int i;
for(i=0; i<N; i++)
  insere_ArvBin(raiz, dados[i]);
.
.
if (remove_ArvBin(raiz, 20))
  printf("20 foi excluido\n");
```



ArvoreBinaria.c: remoção de nó

```
121 struct NO* remove_atual(struct NO* atual) {
122   struct NO *no1, *no2;
123
124   // Sem filho da esquerda. Apontar para o filho da
      direita (trata nó folha e nó com 1 filho).
125   if (atual->esq == NULL){
126     no2 = atual->dir;
127     free(atual);
128     return no2;
129   }
130
131   // Procura filho mais à direita na subárvore da
      esquerda.
132   no1 = atual;
133   no2 = atual->esq;
134   while (no2->dir != NULL){
135     no1 = no2;
136     no2 = no2->dir;
137   }
```

```
138
139   // Copia o filho mais à direita na subárvore da
      esquerda para o lugar do nó removido.
140   if (no1 != atual){
141     no1->dir = no2->esq;
142     no2->esq = atual->esq;
143   }
144   no2->dir = atual->dir;
145
146   free(atual);
147   return no2;
148 }
149
150 int remove_ArvBin(ArvBin *raiz, int valor){
151   if (raiz == NULL)
152     return 0;
153   struct NO* ant = NULL;
154   struct NO* atual = *raiz;
155   while (atual != NULL){
```



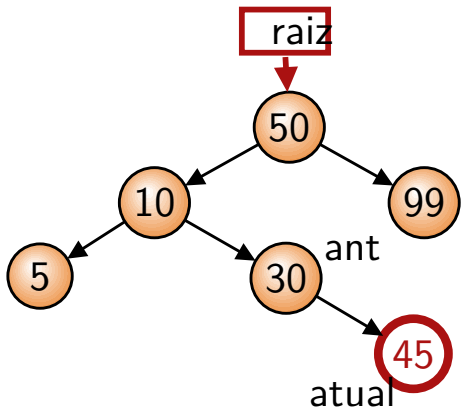
```
156     //Achou o nó a ser removido. Tratar o tipo de remoç
      ão.
157     if (valor == atual->info){
158       if (atual == *raiz)
159         *raiz = remove_atual(atual);
160       else{
161         if (ant->dir == atual)
162           ant->dir = remove_atual(atual);
163         else
164           ant->esq = remove_atual(atual);
165       }
166       return 1;
167     }
168     // Continua buscando na árvore nó a ser removido
169     ant = atual;
170     if (valor > atual->info)
171       atual = atual->dir;
172     else
173       atual = atual->esq;
174   }
```

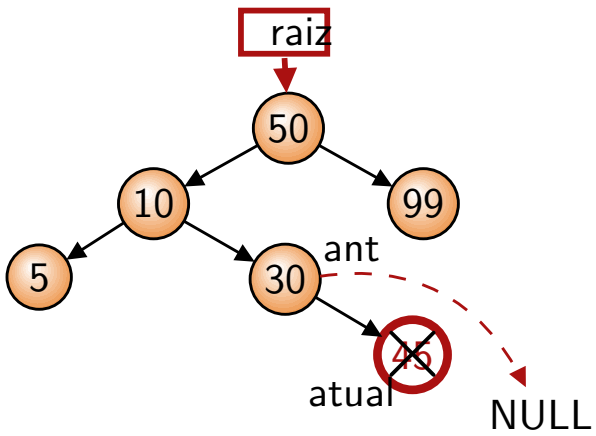
```
175    return 0;
176 }
```



## Removendo um nó folha

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

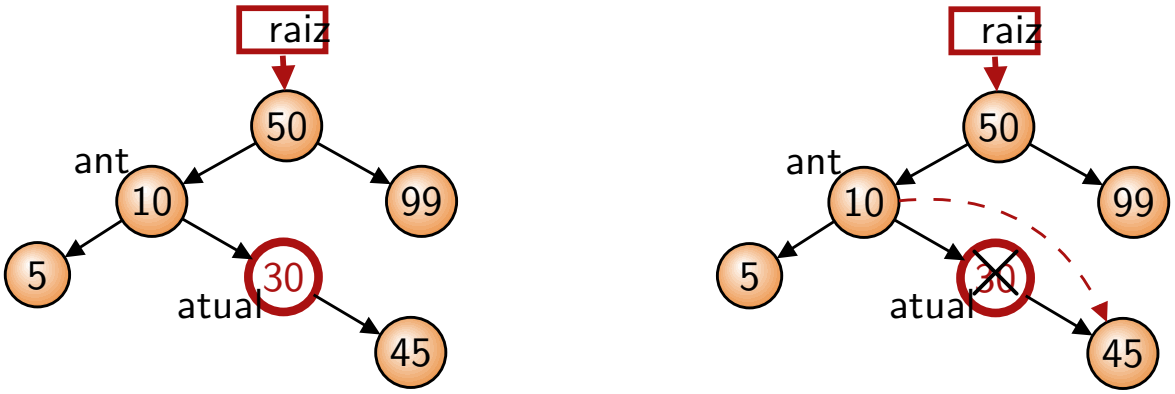



Fonte: BACKES (2023).

## Removendo um nó sem filho à esquerda

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

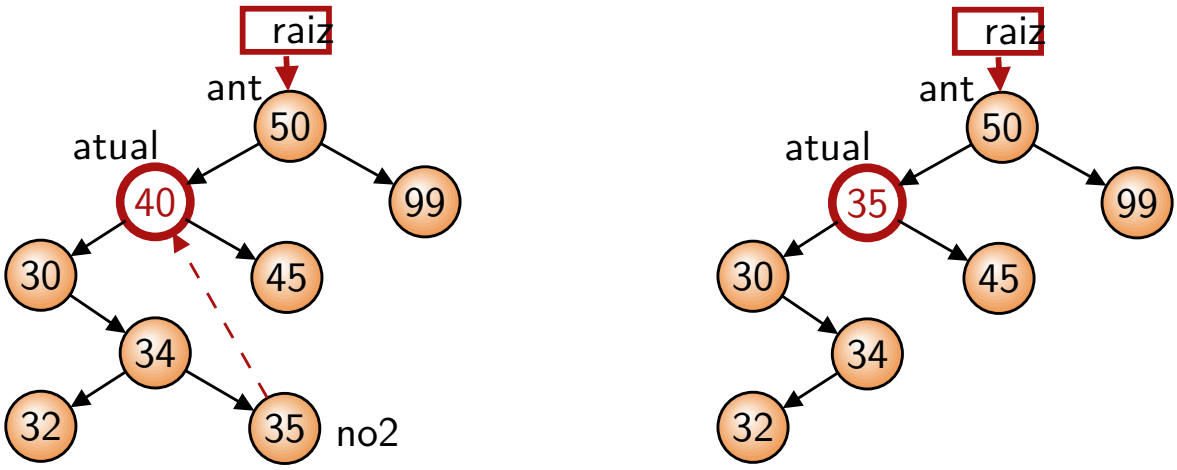



Fonte: BACKES (2023).



## Removendo um nó com dois filhos

- Remove-se o nó "atual" e o substitui pelo nó mais à direita de sua subárvore da esquerda.




Fonte: BACKES (2023).

# Sumário


1. Árvore binária de busca
   - Definição
   - Inserção na ABB
   - Busca na ABB
   - Remoção na ABB
   - Exercícios




# Exercícios I

1. Implemente uma versão não-recursiva do método de inserção de nó em árvore binária de busca.
2. Escreva uma função que retorne se uma árvore é binária de busca ou não.
3. Escreva uma função que encontre o maior valor existente em uma árvore binária de busca.
4. Escreva uma função que encontre o menor valor existente em uma árvore binária de busca.
5. Desenhe a ABB resultante da inserção dos seguintes elementos (na ordem dada):
   1. 60, 22, 64, 33, 67, 11, 75, 13, 8, 65.
   2. 8, 11, 13, 22, 33, 60, 64, 65, 67, 75.

# Referências


1. BACKES, A. R. **Algoritmos e Estruturas de Dados em Linguagem C**. 1. ed. Rio de Janeiro: LTC, 2023. Disponível em: https://www.sistemas.ufu.br/biblioteca-gateway/minhabiblioteca/9788521638315.
2. CORMEN, T. H.; et al. **Algoritmos**. 4. ed. Rio de Janeiro: GEN LTC, 2024. Disponível em: https://www.sistemas.ufu.br/biblioteca-gateway/minhabiblioteca/9788595159914.


Os materiais de parte desta seção foram gentilmente cedidos por André R. Backes (UFSCar); Bruno Travençolo e Christiane Brasil (FACOM/UFU).
TAD árvore binária de busca: disponível em: https://www.facom.ufu.br/~backes/wordpress/ProjArvoreBinaria.zip.
Acesso em: 24/fev/2025.
Adaptações: Renato Pimentel, FACOM/UFU