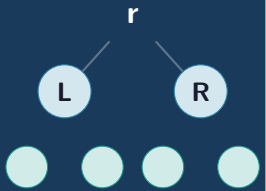
Estruturas de Dados

Aula 9 – Árvores Binárias

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026



Agenda

Definição

Altura

Árvore Binária de Busca

Inserção

Remoção

Percursos

Implementação Completa

Comparativo e Complexidade

Exercícios

Árvores Binárias

Ordem 2 — no máximo dois filhos por nó

O que é uma Árvore Binária?

Definição

Árvore em que cada nó pode ter **zero, um ou dois filhos** (filho esquerdo e filho direito). Cada nó armazena um **valor** e dois ponteiros: `left` e `right`.

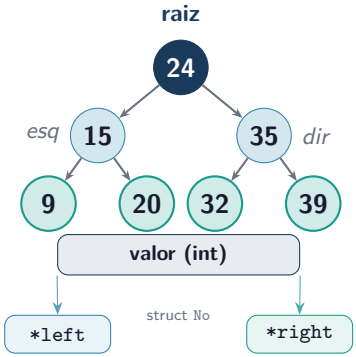
Estrutura recursiva

Uma árvore binária é:

- **Vazia** (`NULL`); ou
- Um **nó raiz** com dois ponteiros para **subárvore esquerda** e **subárvore direita** (cada uma também é uma árvore binária)

Vantagem sobre vetores/listas

Busca em $O(\log N)$ — cada comparação elimina metade dos elementos restantes.



Estrutura do Nó em C

no.h — Definição da struct

```c
typedef struct no {
  int valor;
  struct no *left;
  struct no *right;
} No;
/* Criar um novo nó */
No *novoNo(int v) {
  No *n = malloc(sizeof(No));
  n->valor = v;
  n->left  = NULL;
  n->right = NULL;
  return n;
}
```

Legenda das cores

- **Raiz** da árvore
- Nó **interno**
- Nó **folha** (sem filhos)
- Nó **recém-inserido**
- Nó sendo **removido**

Memória: cada nó ocupa `sizeof(int) + 2×sizeof(No*)` = 4 + 16 = **20 bytes** (64 bits).

Altura da Árvore Binária

O tempo máximo de busca

Altura, Nível e Profundidade

▪ Nível (ou profundidade)

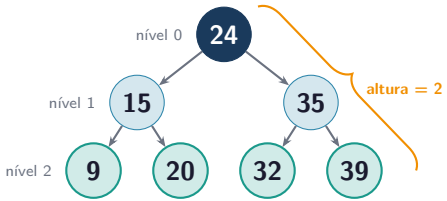
Posição de um nó na hierarquia.
Raiz = nível 0.
Filhos da raiz = nível 1. Etc.

▪ Altura de um nó

Número de *arestas* no maior caminho entre esse nó e um nó-folha descendente.
Folha tem altura **0**.

▪ Altura da árvore

Altura da raiz = altura da árvore.
Árvore vazia: altura $= -1$.
Uma árvore com N nós tem altura **mínima** $= \lfloor \log_2 N \rfloor$ (balanceada).



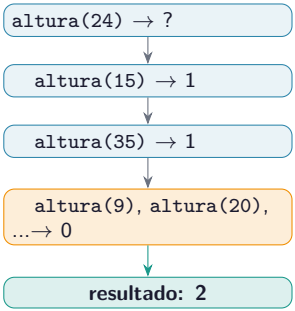
Árvore balanceada com 7 nós:
$\lfloor \log_2 7 \rfloor = 2$ arestas $\Rightarrow$ busca em no máximo 3 comparações.

Calculando a Altura Recursivamente

altura.c

```c
/* Retorna a altura da arvore
   (em numero de arestas).
   Arvore vazia: retorna -1. */
int altura(No *raiz) {
  if (raiz == NULL)
    return -1; // caso base
  int h_esq = altura(raiz->left);
  int h_dir = altura(raiz->right);
  if (h_esq > h_dir)
    return h_esq + 1;
  return h_dir + 1;
}
```

Rastreio na árvore de exemplo:



Complexidade

Visita cada nó exatamente uma vez:
$T(n) = O(n)$, $S(n) = O(h)$ (pilha).

Árvore Binária de Busca (ABB)

Propriedade de ordenação

Propriedade da ABB

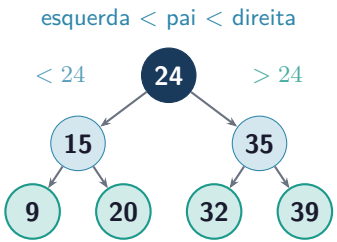
Invariante fundamental

Para **todo** nó N da árvore:

- Todo valor na **subárvore esquerda** $<$ `N.valor`
- Todo valor na **subárvore direita** $>$ `N.valor`
- (sem duplicatas, por convenção)

Busca em $O(\log N)$

A cada passo, descartamos **metade** da árvore. Para $N = 1.000.000$ elementos:
vetor não-ord.: $O(N) = 10^6$ passos
ABB balanceada: $O(\log N) \approx 20$ passos



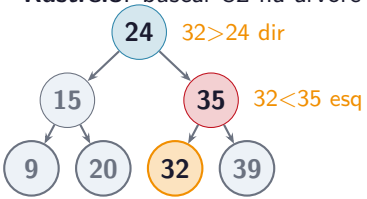
Em-ordem (E→R→D) produz a sequência **ordenada**:
$9, 15, 20, 24, 32, 35, 39$

Busca na ABB

buscar.c

```c
/* Busca 'val' na ABB.
   Retorna o nó ou NULL. */
No *buscar(No *raiz, int val) {
  if (raiz == NULL)
    return NULL;//não achou
  if (val == raiz->valor)
    return raiz;//achou!
  if (val < raiz->valor)
    return buscar(raiz->left,val);//esq
  return buscar(raiz->right,val);//dir
}
```

Rastreio: buscar 32 na árvore



Passos

Comparou apenas **3** nós
de um total de 7. $O(\log 7) = 2{,}8$.

Inserção Ordenada

Mantendo a propriedade da ABB

Inserção na ABB — Algoritmo

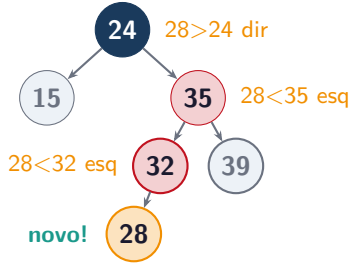
inserir.c

```c
/* Insere 'val' na ABB e
   retorna a nova raiz. */
No *inserir(No *raiz, int val) {
  if (raiz == NULL)
    return novoNo(val); // insere
  if (val < raiz->valor)
    raiz->left =
      inserir(raiz->left, val);
  else if (val > raiz->valor)
    raiz->right =
      inserir(raiz->right, val);
  // val == raiz->valor: ignora
  return raiz;
}
```

Regra

Menor → **esquerda**. Maior → **direita**.
Repete até `NULL` ⇒ cria folha.

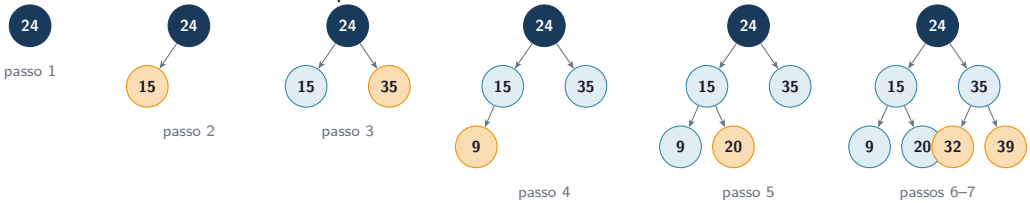
Inserindo 28 na árvore:



Cada inserção leva $O(h)$ passos.
$h = O(\log N)$ se a árvore estiver balanceada.

Inserção — Sequência Passo a Passo

Inserindo a sequência **24, 15, 35, 9, 20, 32, 39** em ordem:



Remoção

Três casos possíveis

Remoção — Os 3 Casos

▪ Caso 1: Nó folha

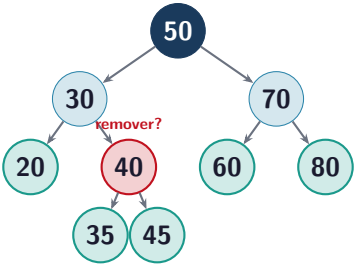
Sem filhos: aponta o ponteiro do pai para `NULL` e libera o nó com `free()`.

▪ Caso 2: Um filho

O pai do nó removido passa a apontar diretamente para o único filho do nó removido.

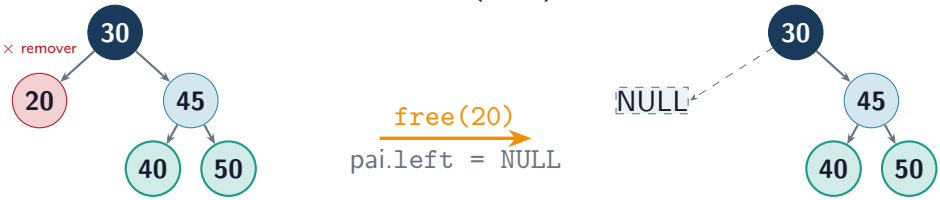
▪ Caso 3: Dois filhos

Substitui pelo **maior da subárvore esquerda** (predecessora) **ou** pelo **menor da subárvore direita** (sucessora).



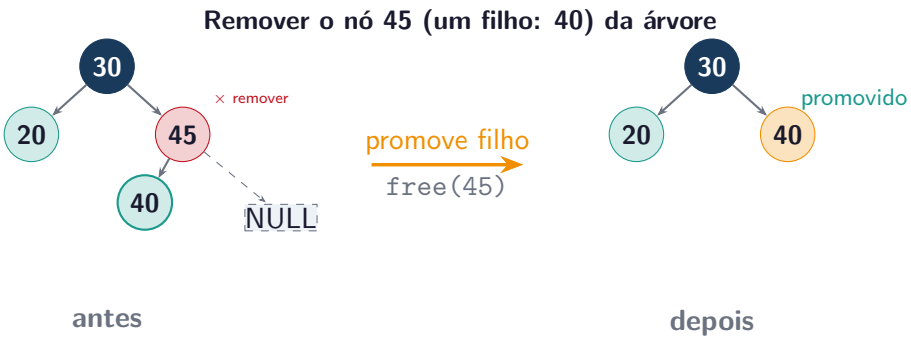
Remoção — Caso 1: Nó Folha

Remover o nó 20 (folha) da árvore



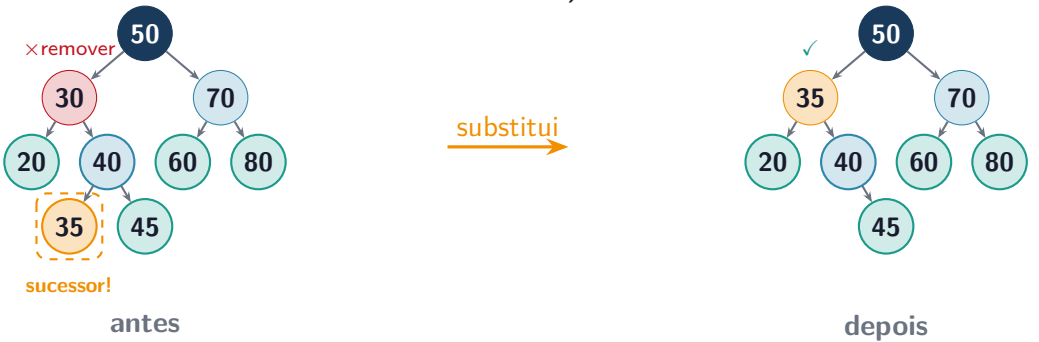
antes **depois**

Remoção — Caso 2: Um Filho



Remoção — Caso 3: Dois Filhos (via sucessor)

Remover o nó 30 (dois filhos) — substituir pelo menor da subárvore direita (sucessor in-ordem)



Observação importante

O **sucessor in-ordem** (o menor da subárvore direita) tem, no máximo, **um filho à direita**. Portanto, sua própria remoção se enquadra no Caso 1 ou no Caso 2. A propriedade da ABB é preservada.

Remoção — Código Completo

remover.c — auxiliar + casos 1–2

```
1  //Retorna o menor nó da subárvore
2  No *menorNo(No *raiz){
3    while (raiz->left != NULL)
4          raiz = raiz->left;
5    return raiz;
6  }
7  No *remover(No *raiz, int val){
8    if (raiz == NULL) return NULL;
9    if (val < raiz->valor)
10     raiz->left =
11         remover(raiz->left,val);
12   else if (val > raiz->valor)
13     raiz->right =
14         remover(raiz->right,val);
```

remover.c — caso 3

```
1    else {// encontrou
2      if (raiz->left == NULL) {
3        No *t = raiz->right;
4        free(raiz);
5        return t; // casos 1,2
6      }
7      //caso 3: substitui pelo sucessor
8      No *suc = menorNo(raiz->right);
9      raiz->valor = suc->valor;
10     raiz->right =
11        remover(raiz->right,suc->valor);
12   }
13   return raiz;
14 }
```

Remoção — Resumo dos 3 Casos

Os 3 casos no código

Caso 1 (folha):
`left==NULL && right==NULL` ⇒ retorna `NULL`

Caso 2 (um filho):
`left==NULL` ⇒ retorna `right`
`right==NULL` ⇒ retorna `left`

Caso 3 (dois filhos):
Copia valor do sucessor, remove o sucessor recursivamente.

Complexidade: $O(h)$ onde h é a altura.

Melhor caso (balanceada): $O(\log N)$.

Pior caso (degenerada): $O(N)$.

Percursos

Pré-Ordem, Em-Ordem e Pós-Ordem

Os 3 Percursos em Árvore Binária

▪ Pré-Ordem (RED)

Raiz → **E**sq → **D**ir

Visita raiz **antes** dos filhos.

Útil para: copiar/serializar, expressões prefixadas.

▪ Em-Ordem (ERD)

Esq → **R**aiz → **D**ir

Visita raiz **entre** os filhos.

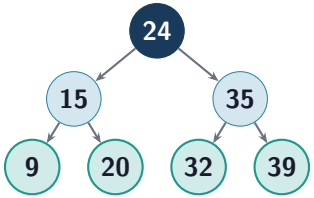
Resulta em sequência **ordenada crescente** (ABB).

▪ Pós-Ordem (EDR)

Esq → **D**ir → **R**aiz

Visita raiz **depois** dos filhos.

Útil para: deletar a árvore, expressões posfixadas.



Pré: **24,15,9,20,35,32,39** Em: **9,15,20,24,32,35,39** Pós: **9,20,15,32,39,35,24**

Percursos — Código dos Três

percursos.c

```c
void preOrdem(No *raiz) {
  if (raiz == NULL) return;
  printf("%d ", raiz->valor);// R
  preOrdem(raiz->left);      // E
  preOrdem(raiz->right);     // D
}
void emOrdem(No *raiz) {
  if (raiz == NULL) return;
  emOrdem(raiz->left);       // E
  printf("%d ", raiz->valor);// R
  emOrdem(raiz->right);      // D
}
void posOrdem(No *raiz) {
  if (raiz == NULL) return;
  posOrdem(raiz->left);      // E
  posOrdem(raiz->right);     // D
  printf("%d ", raiz->valor);// R
}
```

Saídas para a árvore exemplo

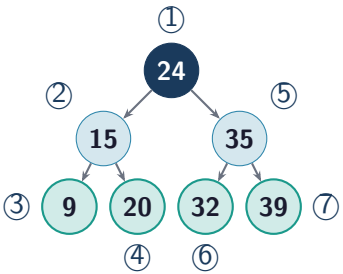
```
Pré-Ordem (RED):
24 15 9 20 35 32 39
Em-Ordem (ERD) -- ORDENADO:
9 15 20 24 32 35 39
Pós-Ordem (EDR):
9 20 15 32 39 35 24
```

Propriedade chave

Em-Ordem em uma ABB sempre produz os elementos em ordem **crescente**. Use para imprimir todos os valores ordenados.

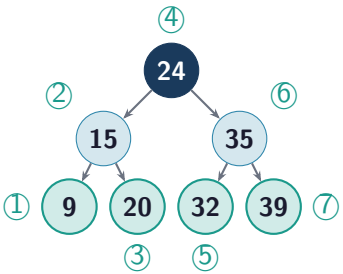
Visualização dos Percursos

Pré-Ordem



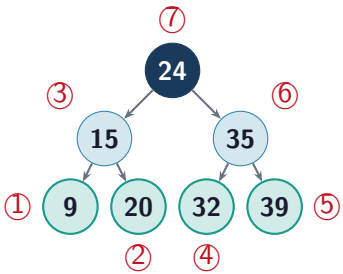
24, 15, 9, 20, 35, 32, 39

Em-Ordem



9, 15, 20, 24, 32, 35, 39

Pós-Ordem



9, 20, 15, 32, 39, 35, 24

Implementação Completa

Imprimir e Menu interativo

Impressão Visual da Árvore

imprimir.c — exibição rotacionada

```c
/* Imprime a árvore rotacionada
   90 graus (direita em cima).
   nível: profundidade do nó  */
void imprimir(No *raiz,int nivel) {
  if (raiz == NULL) return;
  imprimir(raiz->right,nivel + 1);
  for (int i = 0; i < nivel; i++)
    printf("    ");//indenta
  printf("%d\n", raiz->valor);
  imprimir(raiz->left,nivel + 1);
}
/* Chamada: imprimir(raiz, 0); */
```

- **Saída p/ a árvore exemplo**

```
        39
    35
        32
24
        20
    15
        9
```

Como ler

Gire o terminal 90^o no sentido anti-horário: o texto representa a árvore de cima para baixo, com a raiz à **esquerda**.

Menu Interativo

menu.c

```c
void menu(void) {
  printf("\n=== ABB ===\n");
  printf("1. Inserir\n");
  printf("2. Buscar\n");
  printf("3. Remover\n");
  printf("4. Pré-Ordem\n");
  printf("5. Em-Ordem\n");
  printf("6. Pós-Ordem\n");
  printf("7. Imprimir arvore\n");
  printf("8. Altura\n");
  printf("9. Sair\n");
  printf("Opção: ");
}
```

Menu Interativo — main.c

main.c

```c
int main(void) {
  No *raiz = NULL;
  int op, val;
  do{
    menu();
    scanf("%d", &op);
    switch(op) {
      case 1:
        printf("Valor: ");
        scanf("%d", &val);
        raiz = inserir(raiz, val);
        break;
      case 2:
        printf("Valor: ");
        scanf("%d", &val);
        if (buscar(raiz, val))
          printf("Encontrado!\n");
        else
          printf("Não existe!\n");
        break;
```

main.c — continuação

```c
      case 3: printf("Valor: ");
              scanf("%d", &val);
              raiz = remover(raiz,
                   val);
              break;
      case 4: preOrdem(raiz);
              printf("\n"); break;
      case 5: emOrdem(raiz);
              printf("\n"); break;
      case 6: posOrdem(raiz);
              printf("\n"); break;
      case 7: imprimir(raiz, 0);
              break;
      case 8:
        printf("Altura: %d\n",
               altura(raiz));
        break;
    }
  } while (op != 0);
  return 0;}
```

Complexidade

ABB vs. outras estruturas

Resumo de Complexidade

Estrutura	Busca	Inserção	Remoção	Ordenação
Vetor não-ordenado	$O(N)$	$O(1)$	$O(N)$	$O(N \log N)$
Vetor ordenado	$O(\log N)$	$O(N)$	$O(N)$	$O(1)$ (já ordenado)
Lista encadeada	$O(N)$	$O(1)$	$O(1)$	$O(N \log N)$
ABB balanceada	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(N)$
ABB degenerada	$O(N)$	$O(N)$	$O(N)$	$O(N)$

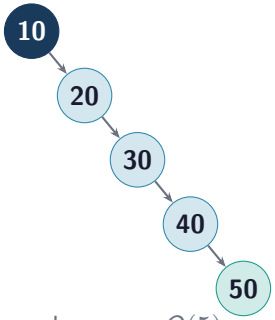
ABB degenerada

Inserir chaves já **ordenadas** (ex.: 1, 2, 3, 4...) degenera a árvore numa **lista encadeada** de altura $N - 1$.
Solução: árvores auto-balanceadas (AVL, Rubro-Negra).

Árvore Degenerada vs. Balanceada

Inserindo 10, 20, 30, 40, 50 em ordem:

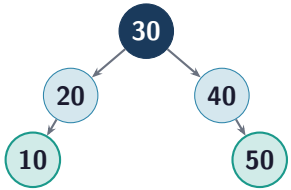
Degenerada ($h = 4$)



busca $= O(5)$

vs.

Balanceada ($h = 2$)



busca $= O(3)$

Próximos passos

Para garantir $O(\log N)$ em todas as operações independente da ordem de inserção: **Árvore AVL** (rotações simples/duplas) ou **Árvore Rubro-Negra** (usada no `std::map` do C++).

Exercícios

Pratique!

Exercício 1 — Construção

Insira a sequência **{40, 20, 60, 10, 30, 50, 70}** em uma ABB inicialmente vazia. Desenhe a árvore resultante e escreva os três percursos: **pré-ordem**, **em-ordem** e **pós-ordem**.

Exercício 2 — Remoção

Na árvore do Exercício 1, remova os nós **10** e depois **40**. Para cada remoção, identifique o caso aplicado (1, 2 ou 3) e desenhe a árvore resultante.

Exercício 3 — Código

Implemente: `int contarFolhas(No *r)`, que retorna o número de nós **folha** (sem filhos).

Dica: folha ⇔ `left == NULL && right == NULL`.

Exercício 4 — Desafio

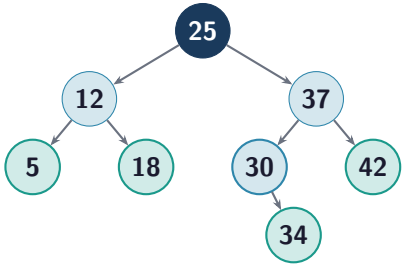
Implemente `int ehABB(No *r, int min, int max)` que verifica se uma árvore binária é uma ABB válida usando os limites $[min, max]$ para cada nó. Como chamá-la inicialmente?

Exercício — Trace na Árvore

Dada a **ABB ao lado**, responda:

Questões

1. Escreva o percurso **em-ordem**.
2. Qual a **altura** da árvore?
3. Remova o nó **37**. Qual caso? Qual é o seu sucessor in-ordem?
4. Insira **28**. Desenhe a posição final.



Dica

Em-ordem: percorra E → R → D e anote. Altura: maior caminho raiz → folha em arestas.

Gabarito (em-ordem):
5, 12, 18, 25, 30, 34, 37, 42

Altura: 3 **Sucessor de 37:** 42

Fim da Aula 9

Dúvidas e próximos passos

Próxima Aula e Referências

Próxima aula

- **Árvores AVL**: balanceamento automático
- Rotações simples e duplas
- Fator de balanceamento
- Aplicações: `std::map` e `std::set` do C++

Referências

- CORMEN et al. *Introduction to Algorithms*, 4.ł ed. Cap. 12.
- TENENBAUM et al. *Estruturas de Dados usando C*. Cap. 5

Resumo

- ABB: busca/inserção/remoção em $O(h)$
- Balanceada: $h = O(\log N)$
- Em-ordem = sequência crescente
- 3 casos de remoção: folha, 1 filho, 2 filhos

Dúvidas?

`carolsoko@ifba.edu.br`

IFBA – Campus Feira de Santana