Estruturas de Dados

Aula 8 — Árvores - Fundamentos

Prof. Ana Carolina Sokolonski

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

2026

Agenda

Introdução às Árvores

Terminologia

Propriedades Especiais

Representação em C

Tipos de Árvores

Exercícios

Fim

Árvores

Estruturas de Dados Hierárquicas

O que são Árvores?

Definição

Árvores são estruturas de dados **não lineares** e versáteis que proporcionam **acesso rápido** aos dados. A diferença fundamental em relação a vetores, listas, filas e pilhas é que árvores são **hierárquicas**.

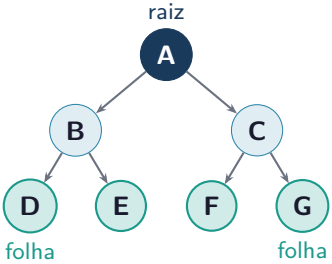
Elementos básicos

1. **Nós** (ou nodos) — armazenam as informações
2. **Arestas** — relacionamentos entre os nós

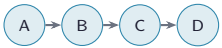
Aplicações

Bancos de dados, sistemas de arquivos, motores de busca, redes sociais, herança em OO, HTML/XML, compiladores.

Exemplo de árvore



Linear vs. Hierarquico

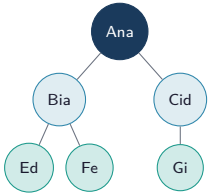


lista (linear)

Aplicações de Árvores

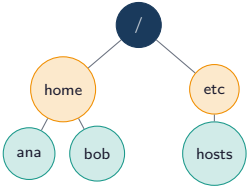
▪ Redes Sociais

Arestas representam graus de amizade ou seguimento.



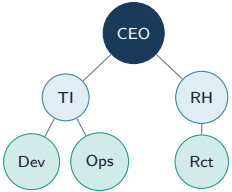
▪ Sistema de Arquivos

Pastas contem arquivos e outras pastas.



▪ Hierarquia Organizacional

Arestas representam subordinação.



Terminologia

Raiz, Folhas, Galhos e mais

Nó Raiz

No início de toda árvore. Único nó **sem pai**. A busca sempre começa nele.

Subárvores

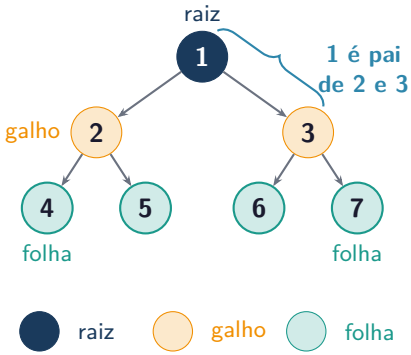
O nó raiz (e qualquer outro nó) pode ter subárvores conectadas. Ao remover a raiz, obtém-se uma **floresta**.

Nós Pai e Filho

Nós superiores que geram outros são **pais**; os gerados são **filhos**. Um nó pode ter vários filhos, mas apenas **um pai**.

Nós Folha

Nós **sem filhos**. Representam o fim de um galho. Grau = 0.



Terminologia — Parte 2

Ancestral e Descendente

Ancestral: nó superior A que se conecta a B por uma sequência de arestas. A é ancestral de B.

Descendente: o inverso — B é descendente de A.

Todo nó é ancestral e descendente de si mesmo.

Ordem da árvore

Grau máximo entre todos os nós. Define o número máximo de filhos por nó.
Árvore binária = ordem 2.

Caminho

Sequência única de arestas que liga dois nós. Em uma árvore, existe exatamente **um** caminho entre quaisquer dois nós.

Nível

Fatia horizontal da árvore. A raiz está no **nível 0**; seus filhos no nível 1, e assim por diante. Nível = profundidade.

Altura

Nível mais alto existente na árvore. Distância da raiz até a folha mais distante. Árvore com apenas a raiz tem altura 0.

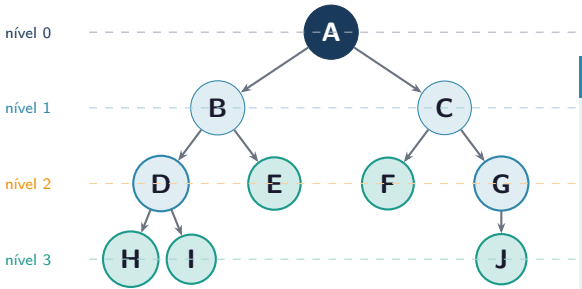
Grau de um Nó

Quantidade de **filhos diretos** do nó. Nós folha têm grau 0.

Profundidade

Distância de um nó até a **raiz** (contando arestas). Raiz tem profundidade 0.

Niveis, Altura e Profundidade — Visualização



Caminho A a I

$A \to B \to D \to I$
Comprimento: 3 arestas

Ancestrais de J

G, C, A (raiz)

Leitura da árvore

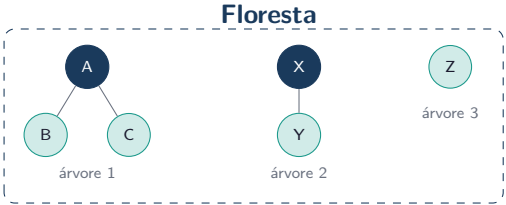
Nível 0:	A (raiz)
Nível 1:	B, C
Nível 2:	D, E, F, G
Nível 3:	H, I, J (folhas)
Altura:	3
Prof.(D):	2
Prof.(H):	3
Grau de B:	2
Grau de G:	1
Ordem:	2

Propriedades Especiais

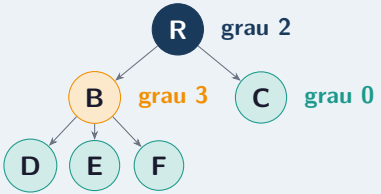
Floresta, Grau e Ordem

Floresta

Conjunto de zero ou mais árvores com nós distintos. Ao remover a raiz de uma árvore, suas subárvores formam uma **floresta**.

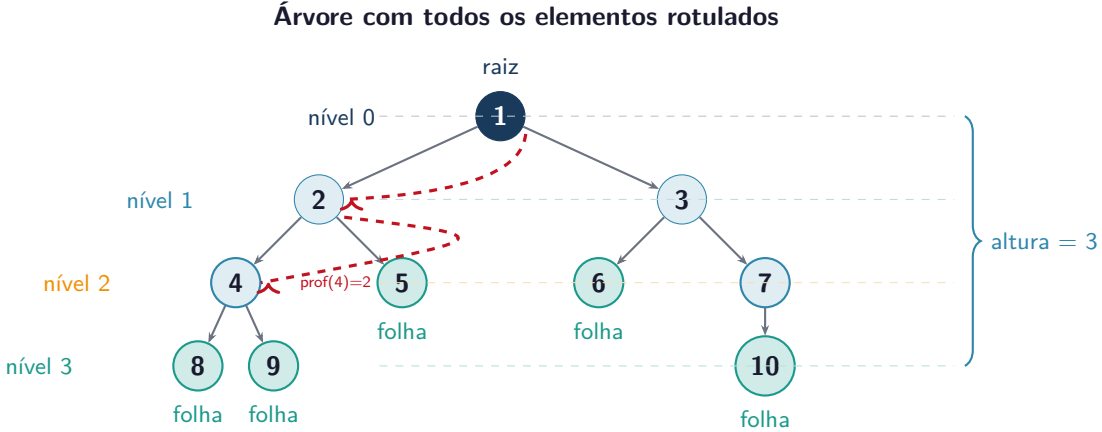


Grau e Ordem



Grau de R: 2
Grau de B: 3 (máximo)
Grau de C, D, E, F: 0
Ordem: **3**

Estrutura Completa de uma Árvore — Referencia Visual



Níveis e Profundidade — Comparação

Profundidade vs. Altura

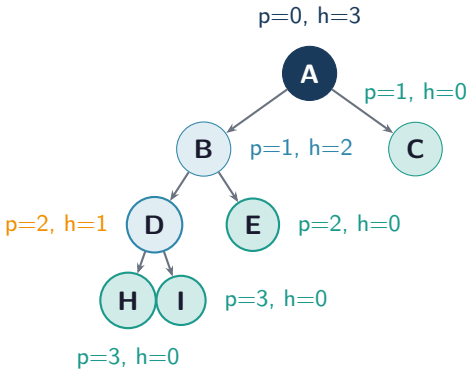
Profundidade e uma propriedade **do nó**: distância da raiz até esse no.

Altura é uma propriedade **da árvore** (ou de um nó): distância do nó até a folha mais distante em sua subárvore.

Altura da árvore = profundidade máxima.

Exemplos numéricos

Nó	Profundidade	Altura do no
A (raiz)	0	3
B	1	2
D	2	1
H	3	0



p = profundidade, h = altura do nó

Representação em C

Struct e operações básicas

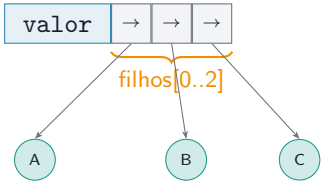
Representação de Árvore em C — Struct

```c
//Nó de árvore genérica (N filhos)
typedef struct No{
    int conteudo;
    int numFilhos;
    struct No **filhos;//array de ponteiros
} No;
No* criarNo(int val, int maxFilhos){
    No *novo = (No*) malloc(sizeof(No));
    novo->conteudo  = val;
    novo->numFilhos = 0;
    novo->filhos=
     (No**)malloc(maxFilhos*sizeof(No*));
    for(int i=0; i<maxFilhos; i++)
        novo->filhos[i]=NULL;
    return novo;
}
//Adiciona filho a um nó
void addFilho(No *pai, No *filho,int pos) {
    pai->filhos[pos] = filho;
    pai->numFilhos++;
}
```

▪ Representação

Cada nó armazena um **array de ponteiros** para seus filhos. O tamanho do array depende da **ordem** da árvore.

Nó com 3 filhos



Percurso em árvore — Pré-ordem e Pós-ordem

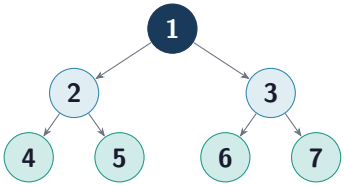
Pré-ordem (raiz, esquerda, direita)

```c
//Visita: raiz -> filhos
void preOrdem(No *no) {
    if (no == NULL) return;
    printf("%d ", no->conteudo);//raiz
    for (int i = 0; i < no->numFilhos;
        i++)
        preOrdem(no->filhos[i]);}
/*Para árvore {1,{2,4,5},{3,6,7}}:
   Saída: 1 2 4 5 3 6 7 */
```

Pós-ordem (filhos, raiz)

```c
/* Visita: filhos -> raiz */
void posOrdem(No *no) {
    if (no == NULL) return;
    for(int i=0; i<no->numFilhos; i++)
        posOrdem(no->filhos[i]);
    printf("%d ",no->conteudo);//raiz
}
/*Para árvore {1,{2,4,5},{3,6,7}}:
   Saída: 4 5 2 6 7 3 1 */
```

Árvore de exemplo e ordens de percurso



Pré-ordem: **1 2 4 5 3 6 7**
Pós-ordem: **4 5 2 6 7 3 1**
Nível (BFS): **1 2 3 4 5 6 7**

Altura e Contagem de Nós

Calcular altura da árvore

```
1  int altura(No *no){
2    if (no == NULL) return -1;
3    if (no->numFilhos == 0) return 0;
4    int maxAltura = -1;
5    for(int i=0; i<no->numFilhos; i++){
6      int h = altura(no->filhos[i]);
7      if (h>maxAltura) maxAltura=h;
8    }
9    return maxAltura + 1;
10 }
11 //árvore {1,{2,4,5},{3,6}}->altura = 2
```

Contar nos totais

```
1  int contarNos(No *no){
2    if (no == NULL) return 0;
3    int total = 1;//conta o próprio nó
4    for (int i=0; i<no->numFilhos; i++)
5      total += contarNos(no->filhos[i]);
6    return total;
7  }
8
9  int contarFolhas(No *no) {
10   if (no == NULL) return 0;
11   if (no->numFilhos == 0) return 1;
12   int total = 0;
13   for (int i=0; i<no->numFilhos; i++)
14     total += contarFolhas(no->filhos[i]);
15   return total;
16 }
```

Tipos de Árvores

Binária, AVL, B, B+ e mais

Principais Tipos de Árvores

▪ Árvore Binária

Cada nó tem **,no máximo, 2 filhos** (esquerdo e direito). Base para BST, AVL, Red-Black.

▪ Árvore Binária de Busca (BST)

Árvore binária onde:
filho esquerda $<$ pai $<$ filho direita
Busca em $O(h)$

▪ Árvore AVL

BST **auto-balanceada**: altura das As subárvores de qualquer nó diferem em, no máximo, 1.
Garante $O(\log n)$ mesmo no pior caso.

▪ Red-Black Tree

BST balanceada usada em `std::map` do C++ e TreeMap do Java.
Altura $\leq 2\log_2(n+1)$.

Principais Tipos de Árvores

▪ Árvore B

Árvore balanceada de ordem $t \geq 2$. Cada nó tem até $2t - 1$ chaves. Projetada para **disco**.
$O(\log_t n)$.

▪ Árvore B+

Variação da B: dados apenas nas folhas, encadeadas. Usada em **SGBDs**.
Excelente para consultas de intervalo.

▪ Trie (Árvore de Prefixos)

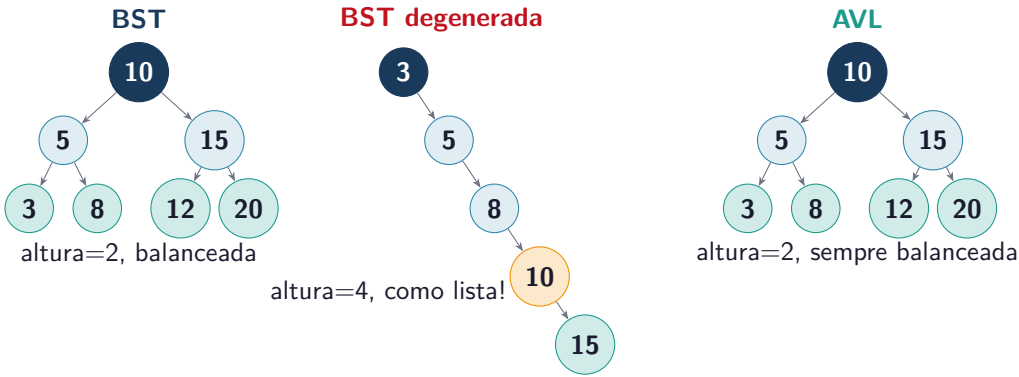
Cada aresta representa um caractere. Busca de palavras em $O(|palavra|)$.
Usada em corretores ortográficos e em autocompletar.

▪ Heap Binário

Árvore binária completa onde o pai é sempre $\geq$ (max-heap) ou $\leq$ (min-heap) que os filhos.
Base do Heap Sort.

Comparação Visual dos Tipos

Mesmos dados {10,5,15,3,8,12,20} em diferentes estruturas



Exercícios

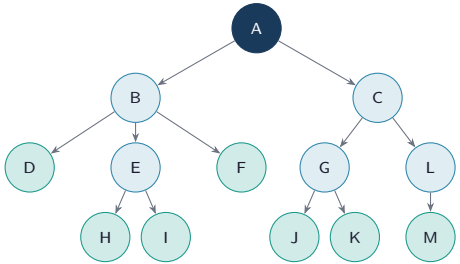
Pratique com Árvores

Exercício 1 — Identificação de Elementos

Enunciado

Para a árvore ao lado, identifique:

1. Qual é a raiz?
2. Quais são as folhas?
3. Qual é a altura da árvore?
4. Qual é a profundidade do nó F?
5. Qual é o grau do nó B?
6. Qual é a ordem da árvore?
7. Liste todos os ancestrais de K.
8. Qual é o caminho da raiz até M?



Gabarito

1. A 2. D,F,H,I,J,K,M 3. 3 4. 2 5. 3
6. 3 7. C,A 8. A→C→L→M

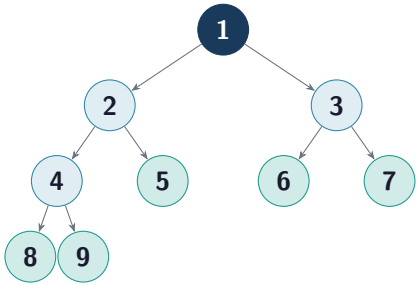
Exercício 2 — Percursos

Enunciado

Para a árvore ao lado, escreva a sequência de nós visitados em:

1. Pré-ordem (raiz, esquerda, direita)
2. Pós-ordem (esquerda, direita, raiz)
3. Percurso por nível (BFS: nível 0, 1, 2, ...)

Além disso: qual é a altura da árvore? Quantos nós possui? Quantas folhas?



Gabarito

Pré: **1 2 4 8 9 5 3 6 7**
Pós: **8 9 4 5 2 6 7 3 1**
BFS: **1 2 3 4 5 6 7 8 9**

Altura: 3, Nós: 9, Folhas: 5

Exercício 3 — Construção

▪ Enunciado A

Desenhe uma árvore que satisfaça TODAS as seguintes condições simultaneamente: 10 nós no total, Altura = 3, Ordem = 3, exatamente 5 folhas e a raiz tem exatamente 2 filhos.

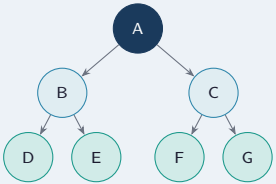
▪ Enunciado B

Dado o seguinte percurso em pré-ordem:
A B D E C F G
e em pós-ordem:
D E B F G C A
Reconstrua a árvore original. *(**dica:** em pré-ordem, o primeiro elemento é sempre a raiz)*

▪ Gabarito B

Pré: A B D E C F G
Pós: D E B F G C A

Raiz = A
Subárvore esquerda com raiz B: {D,E} abaixo
Subárvore direita com raiz C: {F,G} abaixo



Exercício 4 — Implementação

Enunciado

Complete as funções abaixo. Use a struct `No` definido anteriormente.

```c
/* (A) Retorna a profundidade do nó que
     contém val, ou -1 se não existe.
     prof é a profundidade atual */
int buscarProf(No *no, int val, int
    prof) {
    if (no == NULL) return /*(i)*/;
    if (no->conteudo == val)
        return /*(ii)*/;
    for (int i=0; i < no->numFilhos; i
        ++) {
        int r = buscarProf(
                  no->filhos[i], val,
                  /*(iii)*/);
        if (r != -1) return r;
    }
    return -1;
}
```

```c
//(B) Retorna 1 se 'val' é uma folha
int eFolha(No *no, int val) {
    if (no == NULL) return 0;
    if (no->conteudo == val)
        return /* (iv) */;
    for (int i=0; i < no->numFilhos;
         i++) {
        if (eFolha(no->filhos[i],
            val))
            return 1;
    }
    return 0;
}
```

Exercício 4 — Implementação

Gabarito

(i) `-1`
(ii) `prof`
(iii) `prof + 1`
(iv) `(no->numFilhos == 0)`

Chamada de exemplo:

```
buscarProf(raiz, 7, 0)
```

Para a árvore da aula:
$buscarProf(raiz, D, 0) = 2$
$buscarProf(raiz, A, 0) = 0$
$buscarProf(raiz, Z, 0) = -1$

Desafio

Implemente `imprimirCaminho(raiz, val)` que imprime a sequência de nós do caminho da raiz até o nó com valor `val`.

Fim