Árvores – Introdução

Prof. Renato Pimentel

2025/2

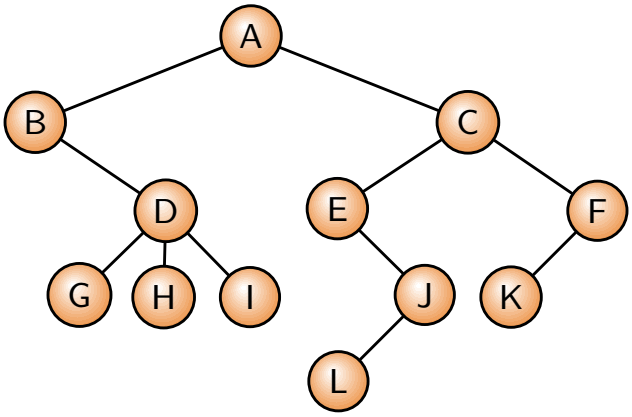
Sumário

1 Introdução a árvores
- Definições básicas
- Árvores binárias
- Representações
- Um tipo abstrato de dados (TAD) para árvore binária
- Exercícios

Árvore

Uma **árvore** é uma abstração matemática usada para representar estruturas hierárquicas não lineares dos objetos modelados.

- É um tipo especial de grafo:
  - Definida usando um conjunto de nós (ou vértices) e arestas
  - Qualquer par de vértices está conectado a apenas uma aresta
    - ★ Grafo não direcionado, conexo e **acíclico** (sem ciclos).



Exemplos de aplicações

1. **Genealogia**: Relações de descendência (pai, filho, etc.)
2. **Gestão/administração**: diagrama hierárquico de uma organização; campeonatos de modalidades desportivas;
3. **Biologia**: taxonomia.
4. **Computação**:
   - **Árvores *trie*** (ou árvores de prefixo): usadas em mecanismos de busca e corretores ortográficos.
   - Representação do **espaço de soluções** de um problema.
     - ★ Ex.: jogo de xadrez – cada elemento da árvore correspondente a uma determinada posição das peças no tabuleiro;
   - Modelagem de algoritmos:
     - ★ Algoritmos e busca em largura (BFS) e profundidade (DFS), árvores de decisão (IA), etc.

Árvore de decisão

Devemos jogar tênis hoje ou não?

```
                (Como está o tempo?)
              /          |          \
      Ensolarado    Nublado    Chuvoso
         /             |          \
  (Umidade?)         Sim      (Vento?)
   /       \                   /      \
Alta      Normal           Fraco     Forte
 |           |               |          |
Não        Sim             Sim        Não
```

Fonte: adaptado de HEMMAT, HAFID (2016).
https://arxiv.org/abs/1611.10338

Estrutura de pastas e arquivos do S. O.

Fonte: FrancoGG - Obra do próprio, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=766938

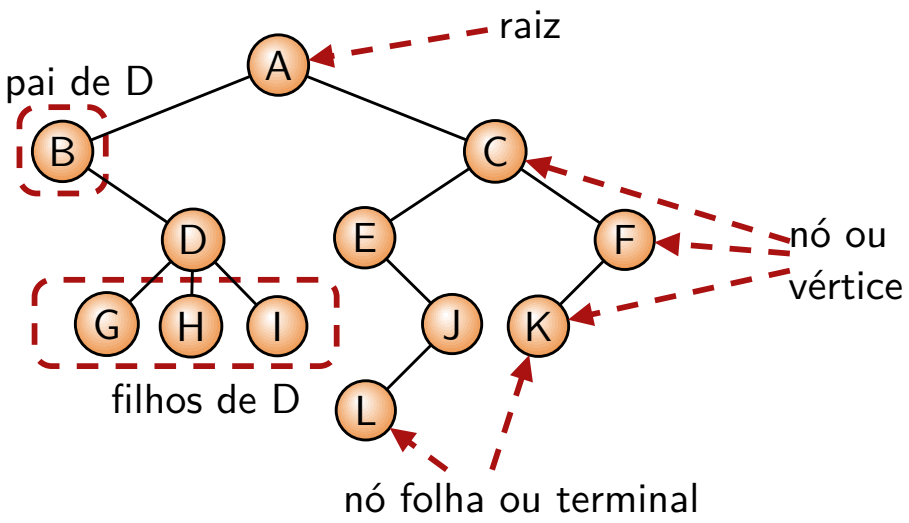
Conceitos

- **Vértice**, ou **nó**, é cada uma das entidades representadas na árvore.
- A representação do vértice depende do problema estudado:
  - Organização do sistema operacional, onde cada vértice representa justamente uma pasta ou diretório do sistema.
  - Árvore *trie*: cada vértice pode representar uma letra ou prefixo.

```
            (C)
           /   \
        (A)    (Ã)
       /   \      \
    (S)    (R)    (O)
    /        |
  (A)        (R)
   |          |
  ( )        (O)
```

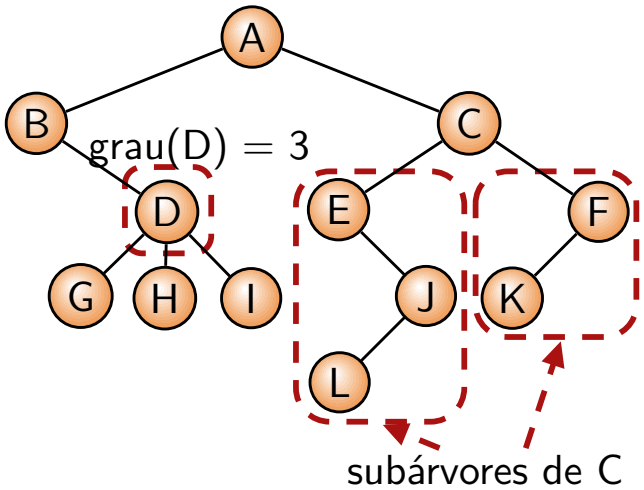
Fonte: https://chatgpt.com

- Assim como em grafos, cada **aresta** representa a ligação (conexão) existente entre dois vértices.
- **Raiz**: vértice mais alto na árvore, o único que não possui pai.
- **Pai** ou **ancestral**: vértice antecessor imediato de outro vértice.
- **Filho**: o vértice sucessor imediato de outro vértice.
- **Nó folha** ou **terminal**: qualquer vértice que não possui filhos.
- **Nó interno** ou **não-terminal**: vértice que possui ao menos um filho.
- **Caminho**: sequência de nós, de modo que sempre existe uma aresta conectando o nó anterior ao seguinte.
  - Existe exatamente um caminho da **raiz** até qualquer um dos vértices.



Subárvores e grau

- Dado um nó, cada filho é considerado a raiz de uma nova **subárvore**;
- Qualquer nó é raiz de uma subárvore que consiste do próprio e dos nós abaixo dele (conceito recursivo).



- O **grau** de um nó é o número de subárvores (ou filhos) que possui.
  - Nós folhas possuem grau zero.

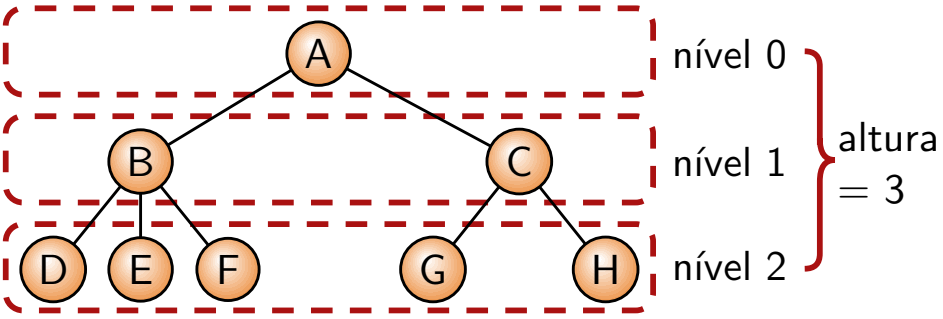
Nível e altura

Nível

- É dado pelo número de nós no caminho entre o nó em questão e a raiz (**nível 0**).
- Nós são classificados em diferentes níveis.

Altura

- Ou **profundidade**, corresponde ao total de níveis de uma árvore;
- Comprimento do **caminho mais longo** da raiz até uma de suas folhas.

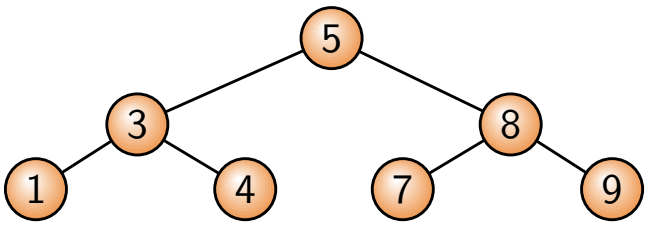


Tipos de árvores

- Na computação, assim como na natureza, existem vários tipos diferentes de árvores.
- Cada uma delas foi desenvolvida pensando-se em distintas possibilidades de aplicações:
  1. árvore binária de busca;
  2. árvore AVL;
  3. árvore rubro-negra;
  4. árvore B, B+ e B*;
  5. árvore 2-3;
  6. árvore 2-3-4;
  7. quadtree;
  8. octree.

Árvore binária de busca

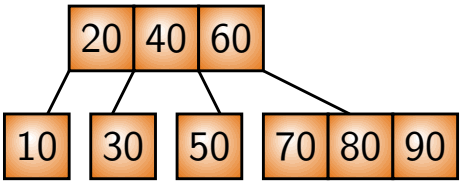
- **Árvore binária** onde, para cada nó, seus descendentes da subárvore da esquerda (direita) possuem valores menores (maiores) que o mesmo.



- Útil em diversas aplicações, como bancos de dados (eficiência de índices de pesquisa), compiladores (análise de símbolos), IA (tomadas de decisão), dicionários, filtragem de informações (*firewall*), etc.
- Dependendo das operações de inserção/remoção, árvore pode ficar **desbalanceada**, prejudicando eficiência. Duas variantes sanam o problema:
  1. Árvore AVL;
  2. Árvore rubro-negra (ou *red-black*).

Árvore B e variantes

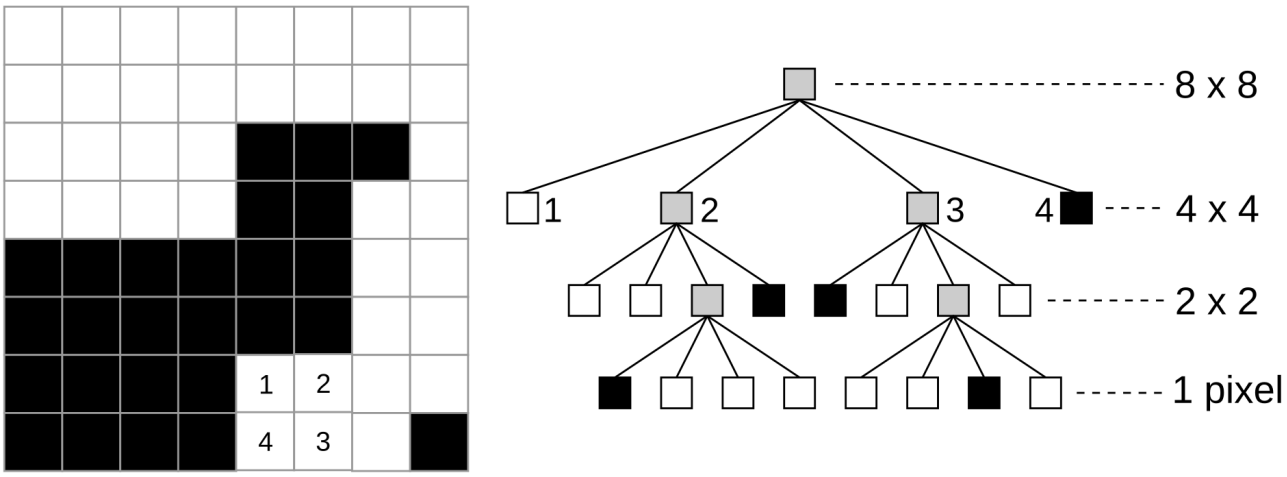
- A árvore B e suas variantes (B+ e B*) são EDs balanceadas usadas principalmente para armazenar e organizar grandes volumes de dados em sistemas que requerem acesso eficiente, como bancos de dados e sistemas de arquivos.



- Evoluções das árvores binárias de busca, otimizadas para minimizar acessos a disco e maximizar eficiência em leitura e escrita.
- **Árvore B+**: dados armazenados somente nas folhas;
- **Árvore B***: otimização para muitas inserções/remoções (política de divisão);
- **Árvores 2-3** e **2-3-4**: Árvores B onde cada nó pode ter 1 ou 2 chaves (e 2 ou 3 filhos); ou 1 a 3 chaves e 2 a 4 filhos, respectivamente.

- **Quadtree**: ED usada para armazenar e organizar informações espaciais de forma eficiente. Particiona recursivamente um espaço 2D em quatro regiões menores, facilitando operações como busca, inserção e remoção de elementos.
- Os dados armazenados em cada folha dependem da aplicação. Mais comuns: computação gráfica (armazenamento, compressão e renderização de imagens), Sistemas de Informações Geográficas (GIS), simulações físicas e jogos (colisões), indexação espacial (buscas), visão computacional.
- **Octree**: equivalente 3D da quadtree.

Fonte: Lkjhsdfljsd - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=47469171



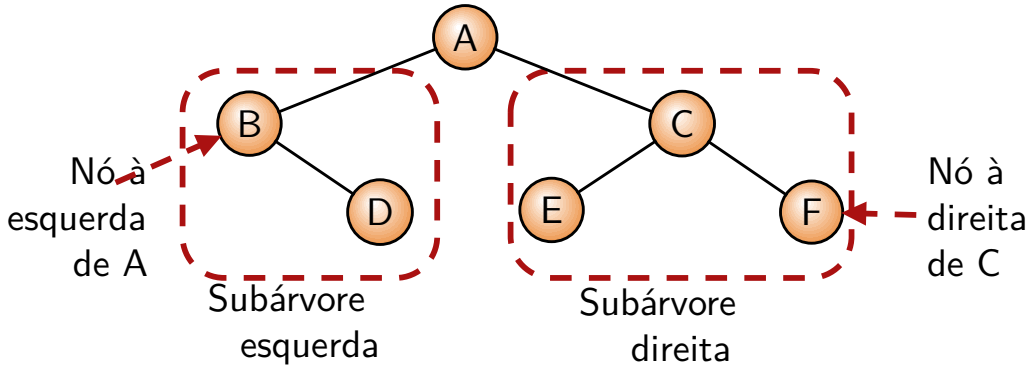
Fonte: Wojciech Muła - Own work, Public Domain,
https://commons.wikimedia.org/w/index.php?curid=4878903

Sumário

1 Introdução a árvores
- Definições básicas
- Árvores binárias
- Representações
- Um tipo abstrato de dados (TAD) para árvore binária
- Exercícios

Árvores binárias

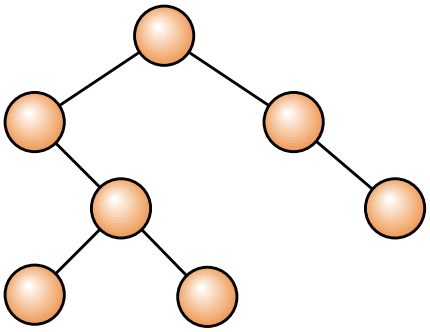
- Árvores onde cada nó pode possuir nenhuma, uma ou **no máximo duas subárvores**:
  - Subárvore da **esquerda** e a da **direita**.
- Úteis em situações onde a cada passo é preciso tomar uma decisão entre duas direções.
  - Ex.: bancos de dados, compiladores, compressão de dados (**algoritmo de Huffman**), representação de expressões aritméticas, etc.



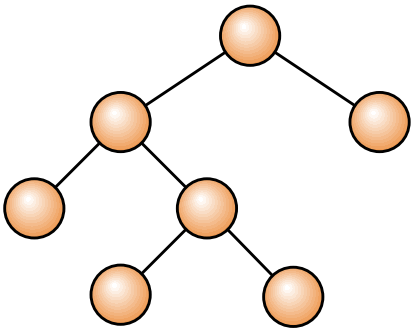
- Existem três tipos de árvores binárias:
  1. Estritamente binária;
  2. Quase completa;
  3. Cheia.

Árvore estritamente binária

- Cada nó possui sempre ou zero (no caso de nó folha) ou duas subárvores;
- Nenhum nó tem **filho único**.



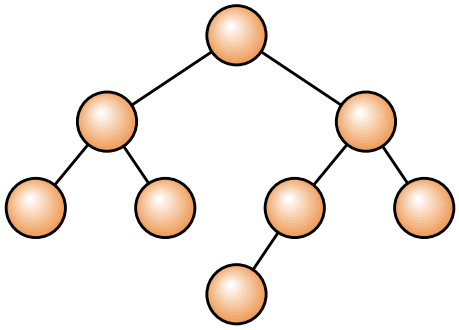
Árvore binária



Árvore estritamente binária

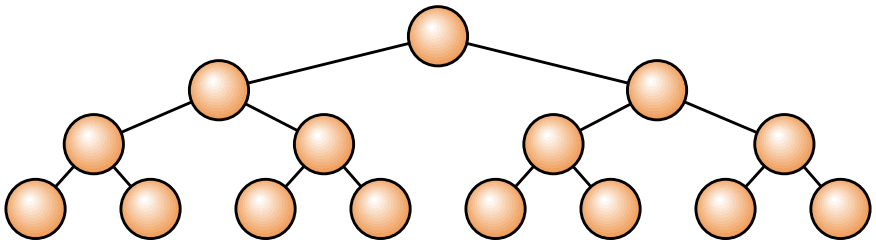
Árvore binária quase completa

- A diferença de altura entre as subárvores de qualquer nó é no máximo 1.
- Se a altura da árvore é H, então cada nó folha está no nível $H - 1$ (último) ou $H - 2$.



Árvore binária cheia ou completa

- Árvore estritamente binária onde todos os nós folhas estão no mesmo nível.



- Características:
  - Há exatamente 2^n nós no nível n;
  - Uma árvore de altura H possui $2^H - 1$ nós.

Sumário

1 Introdução a árvores
- Definições básicas
- Árvores binárias
- Representações
- Um tipo abstrato de dados (TAD) para árvore binária
- Exercícios

- Duas abordagens principais:
  - Usando um *array* – **Alocação estática**;
  - Usando uma lista encadeada – **Alocação dinâmica**.
- Assim como em grafos, a melhor representação, dentre as duas, depende da aplicação. Não existe uma ideal para todos os casos.

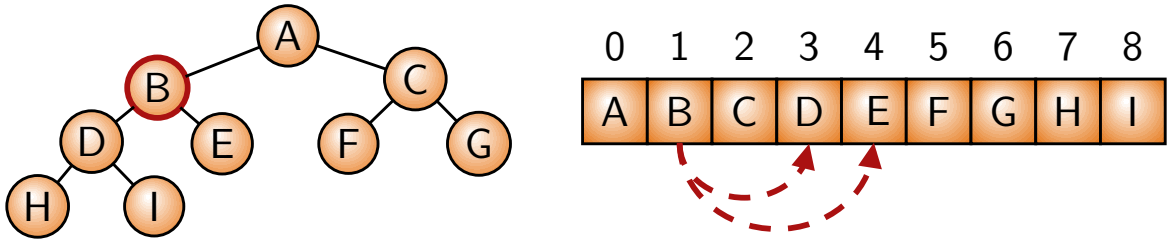
- Independentemente do tipo de alocação usado na implementação da árvore, as seguintes operações básicas são possíveis:
  1. Criação da árvore;
  2. inserção de um elemento;
  3. remoção de um elemento;
  4. busca por um elemento;
  5. destruição da árvore;
  6. informações como número de nós, altura ou se está vazia.

Usando um *array* (*heap*)

- **Alocação estática**: há um máximo de nós – tamanho do *array*.
- A ***heap*** permite uma árvore binária completa ou quase completa.
- Usa 2 funções para a posição dos filhos à esquerda e à direita:

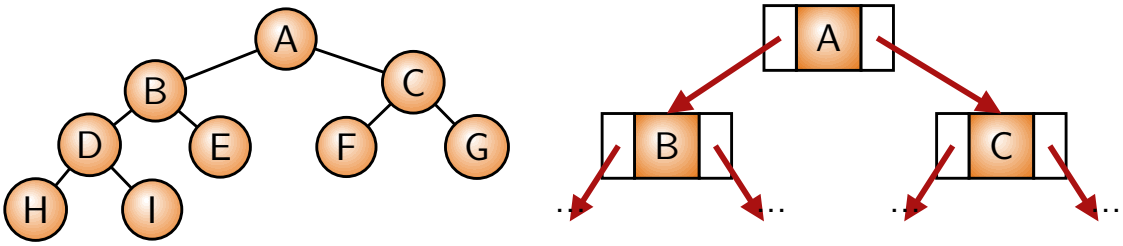
```
FILHO_ESQ(PAI) = 2 * PAI + 1
FILHO_DIR(PAI) = 2 * PAI + 2
```

- Uso: quando todos os elementos e a estrutura da árvore são conhecidos *a priori*. Mais adequado para **buscas** frequentes.



Usando lista encadeada (alocação dinâmica)

- Espaço de memória alocado em tempo de execução;
- Cresce à medida que novos elementos são armazenados, e diminui à medida que elementos são removidos.
- Uso: não necessita que se conheça previamente todos os elementos a serem inseridos na árvore. A árvore a ser representada não precisa ser uma árvore binária completa ou quase completa.
  - Mais adequado na maioria das situações.



Sumário

1 Introdução a árvores
- Definições básicas
- Árvores binárias
- Representações
- Um tipo abstrato de dados (TAD) para árvore binária
- Exercícios

Um tipo abstrato de dados (TAD) para árvore binária

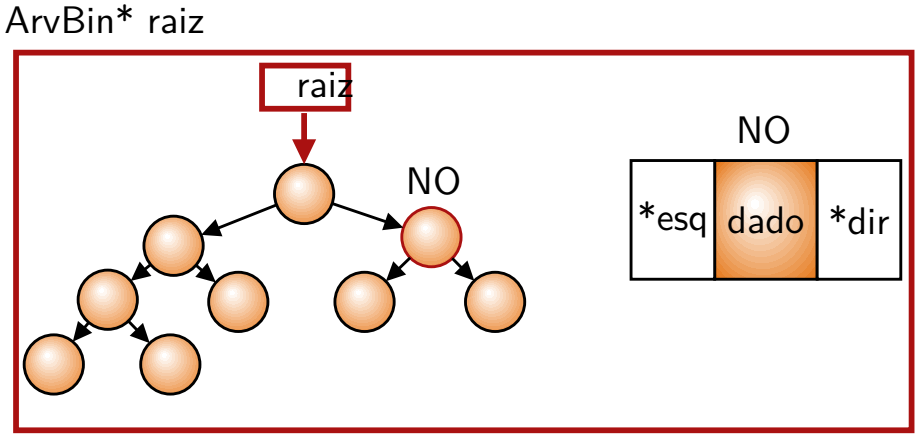
- Consideramos um tipo abstrato de dados **opaco** em Linguagem C para representação **encadeada**.

Relembrando: Tipo opaco

- A ED será referenciada mediante um **ponteiro** para a mesma, ao invés de uma variável do tipo "árvore binária" diretamente.
- Isto permite ocultar a implementação, permitindo sua alteração sem interferir no funcionamento de qualquer código que o utilize – desde que a **interface** seja respeitada.

Alocação dinâmica

- Para guardar o primeiro nó da árvore utilizamos **um ponteiro para ponteiro**;
  - armazena-se o **endereço de um ponteiro**.
- Possibilita mudar quem é a **raiz** da árvore (caso necessário).
  - Mudança não fica "visível" para o usuário da ED.



- Modularização do TAD:
  - **ArvoreBinaria.h**: a **interface** em si: definição do tipo opaco; funções disponíveis para manipular a árvore binária (protótipos).
  - **ArvoreBinaria.c**: o que deve ficar oculto do usuário, além da implementação das funções definidas pela interface. Inclui a definição do **tipo nó**.
- **Tipo nó**: estrutura que contém:
  - `info`: Representa a **informação**, ou seja, os dados correspondentes ao elemento (nó/vértice) em questão;
    - ★ Para fins de simplicidade, será no TAD ilustrado um `int`.
  - `*esq` e `*dir`: ponteiros para os nós filhos à esquerda e à direita do nó.

Interface: ArvoreBinaria.h

ArvoreBinaria.h

```
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);
```

Outras funções serão vistas posteriormente.

Definição do nó: ArvoreBinaria.c

ArvoreBinaria.c: bibliotecas usadas e definição do tipo

```
include <stdio.h>
#include <stdlib.h>
#include "ArvoreBinaria.h" //inclui os protótipos

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

Criando uma árvore binária I

Exemplo de uso

```
ArvBin* raiz = cria_ArvBin();
```

ArvoreBinaria.c: criação da árvore binária

```
11 ArvBin* cria_ArvBin(){
12   ArvBin* raiz = (ArvBin*) malloc(sizeof(ArvBin));
13   if(raiz != NULL)
14     *raiz = NULL;
15   return raiz;
16 }
```


Liberando a árvore binária da memória I

Uso de duas funções:
uma auxiliar (não disponível na interface) percorre e libera os nós; outra – a que o usuário chama – trata a raiz

Exemplo de uso

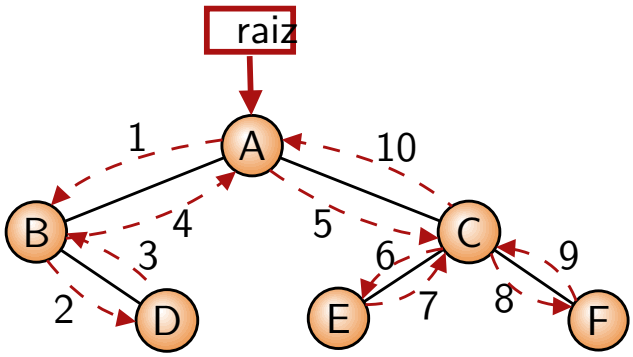
```
ArvBin* raiz = cria_ArvBin();
.
.
libera_ArvBin(raiz);
```

Liberando a árvore binária da memória II

ArvoreBinaria.c: liberando a árvore binária

```
18 void libera_NO(struct NO* no){
19   if(no == NULL)
20     return;
21   libera_NO(no->esq);
22   libera_NO(no->dir);
23   free(no);
24   no = NULL;
25 }
26
27 void libera_ArvBin(ArvBin* raiz){
28   if(raiz == NULL)
29     return;
30   libera_NO(*raiz); // libera cada nó
31   free(raiz); // libera a raiz
32 }
```


Passo-a-passo



	1	visita B
	2	visita D
×	3	libera D, volta p/ B
×	4	libera B, volta p/ A
	5	visita C
	6	visita E
×	7	libera E, volta p/ C
	8	visita F
×	9	libera F, volta p/ C
×	10	libera C, volta p/ A e
×		libera A
		libera raiz

Informações básicas sobre a árvore binária

Algumas informações sobre a árvore estão inclusas no TAD, como:

- A árvore binária está vazia?
- Quantos elementos (nós) possui a árvore?
- Qual a **altura** da árvore?

Verificando se a árvore binária está vazia I

Exemplo de uso

```
ArvBin* raiz = cria_ArvBin();
.
.
if (estaVazia_ArvBin(raiz))
  printf("A árvore binária está vazia\n");
```

ArvoreBinaria.c: árvore está vazia?

```
34 int estaVazia_ArvBin(ArvBin *raiz){
35   if(raiz == NULL)
36     return 1;
37   if(*raiz == NULL)
38     return 1;
39   return 0;
40 }
```

Verificando se a árvore binária está vazia II

- Primeiro teste verifica se ponteiro `ArvBin *raiz` é `NULL` – **problema na criação da árvore**;
- Segundo teste verifica se existe uma raiz (árvore criada com sucesso, mas não possui nenhum nó).

Quantidade de elementos na árvore I

Exemplo de uso

```
ArvBin* raiz = cria_ArvBin();
.
.
printf("A árvore possui %d nós\n", totalNO_ArvBin(raiz)
    );
```

ArvoreBinaria.c: quantidade de nós

```
42 int totalNO_ArvBin(ArvBin *raiz){
43   if (raiz == NULL)
44     return 0;
45   if (*raiz == NULL)
46     return 0;
47   int totalNO_esq = totalNO_ArvBin(&((*raiz)->esq));
48   int totalNO_dir = totalNO_ArvBin(&((*raiz)->dir));
49   return (totalNO_esq + totalNO_dir + 1);
```

Quantidade de elementos na árvore II

```
}
```

- `totalNO_ArvBin` recebe como parâmetro um **ponteiro para ponteiro** – `ArvBin*` ou `struct NO**`;
- Como os nós são ponteiros simples (`struct NO *`), é preciso passar os **endereços** destes nas chamadas recursivas.

Altura da árvore I

Exemplo de uso

```
ArvBin* raiz = cria_ArvBin();
.
.
printf("A árvore tem altura %d\n", altura_ArvBin(raiz))
    ;
```

ArvoreBinaria.c: altura

```
int altura_ArvBin(ArvBin *raiz){
  if (raiz == NULL)
    return 0;
  if (*raiz == NULL)
    return 0;
  int alt_esq = altura_ArvBin(&((*raiz)->esq));
  int alt_dir = altura_ArvBin(&((*raiz)->dir));
  if (alt_esq > alt_dir)
```

Altura da árvore II

```
60      return (alt_esq + 1);
61    else
62      return(alt_dir + 1);
63  }
```

- Como no caso anterior, `altura_ArvBin` recebe como parâmetro um **ponteiro para ponteiro** – `ArvBin*` ou `struct NO**`;
- Como os nós são ponteiros simples (`struct NO *`), é preciso passar os **endereços** destes nas chamadas recursivas.

Sumário

1. Introdução a árvores
   - Definições básicas
   - Árvores binárias
   - Representações
   - Um tipo abstrato de dados (TAD) para árvore binária
   - Exercícios

Exercícios I

1. Escreva uma função que percorra uma árvore binária e conte o seu número de nós.
2. Escreva uma função que percorra uma árvore binária e conte o seu número de nós não-folhas.

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