



DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA

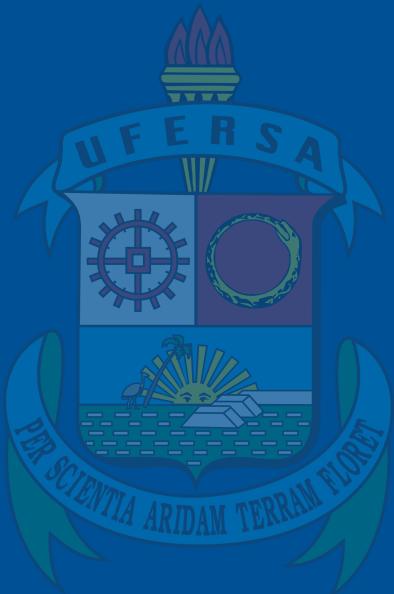
UNIVERSIDADE FEDERAL
UFERSA
RURAL DO SEMI-ARIDO

Árvore Binária

Algoritmos e Estruturas de Dados

2025.2

Agenda



- 1 Definição
- 2 Propriedades
- 3 Falsas Árvores
- 4 Definições (Super) Importantes
- 5 Revisão e Considerações

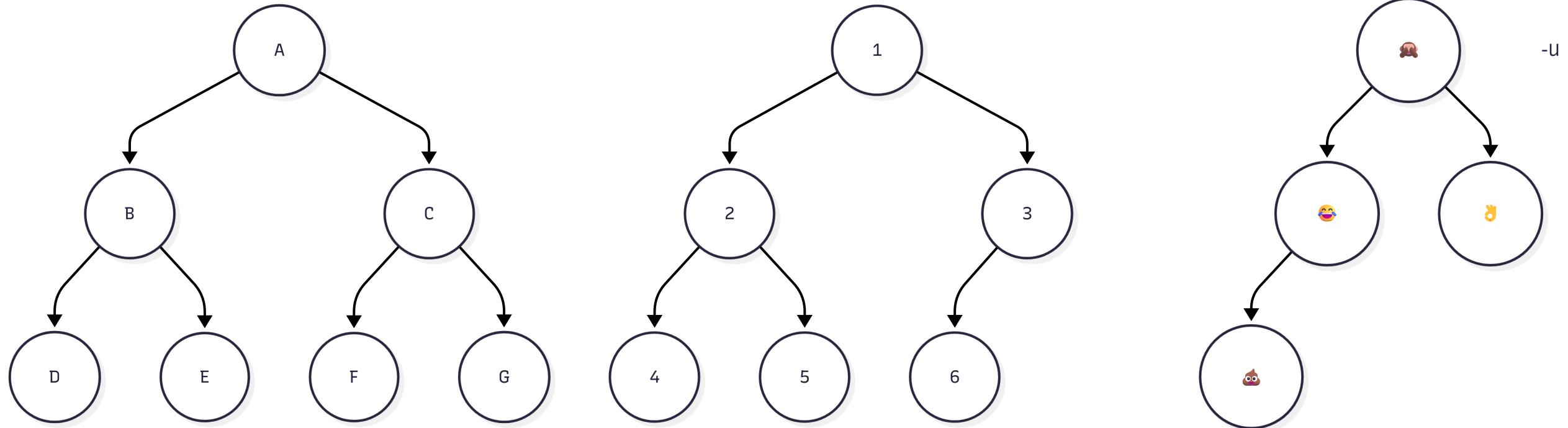
Definição

Definição

Árvore Binária

A árvore binária é uma árvore onde os nós possuem no **máximo dois filhos**. Portanto, um nó qualquer pode possuir 0, 1 ou 2 filhos.

Exemplos



Propriedades

Propriedades das Árvore Binárias

Um árvore binária de altura h tem no máximo $2^{h+1} - 1$ nós

Um árvore binária com n nós tem uma altura mínima de $\lceil \log_2(n + 1) \rceil - 1$

Um árvore binária com n nós tem no máxima $\lceil n/2 \rceil$ nós terminais

Exemplo 1

Uma árvore de altura 4 possuirá, no máximo, $2^{4+1} - 1 = 2^5 - 1 = 31$ nós.

Exemplo 2

Se uma árvore binária possui 7 nós, então sua altura mínima será de

$$\lceil \log_2(7 + 1) \rceil - 1 = \lceil \log_2 8 \rceil - 1 = \lceil 3 \rceil - 1 = 2$$

Além disso, tal árvore possuirá no máximo $\lceil 7/2 \rceil$ nós terminais, ou seja, 4 folhas.

Classificação

Classificação

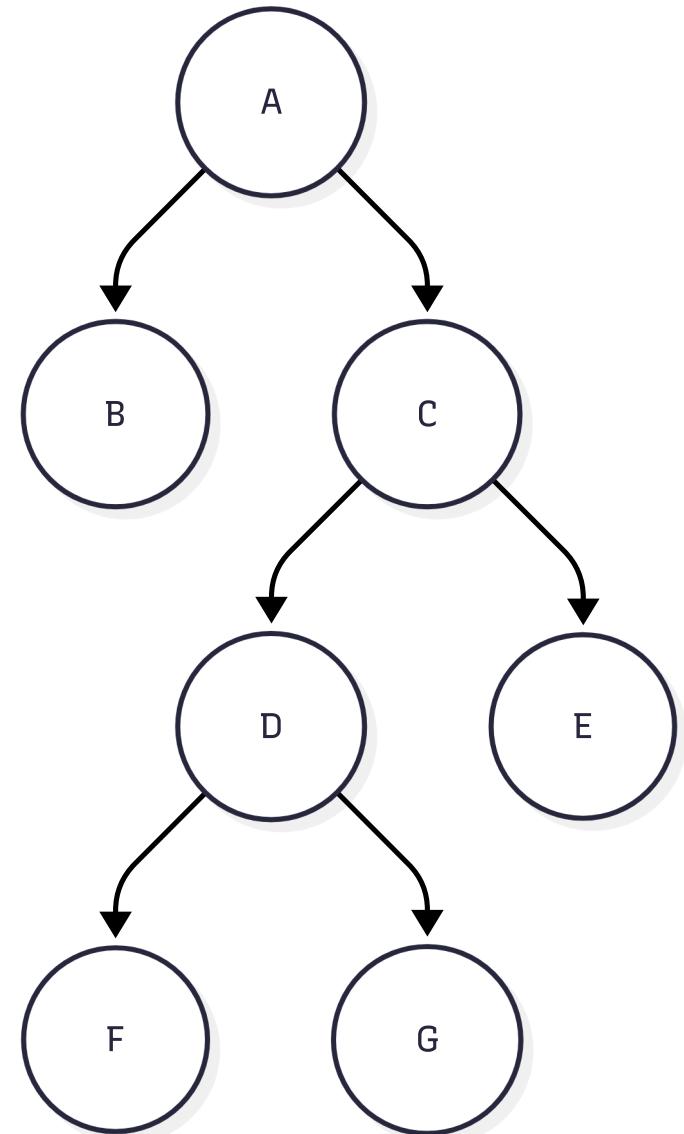
Uma árvore binária pode ser classificada em

- **Cheia,**
- **Completa,**
- **Perfeita** ou
- **Degenerada,**

dependendo do arranjo de seus nós.

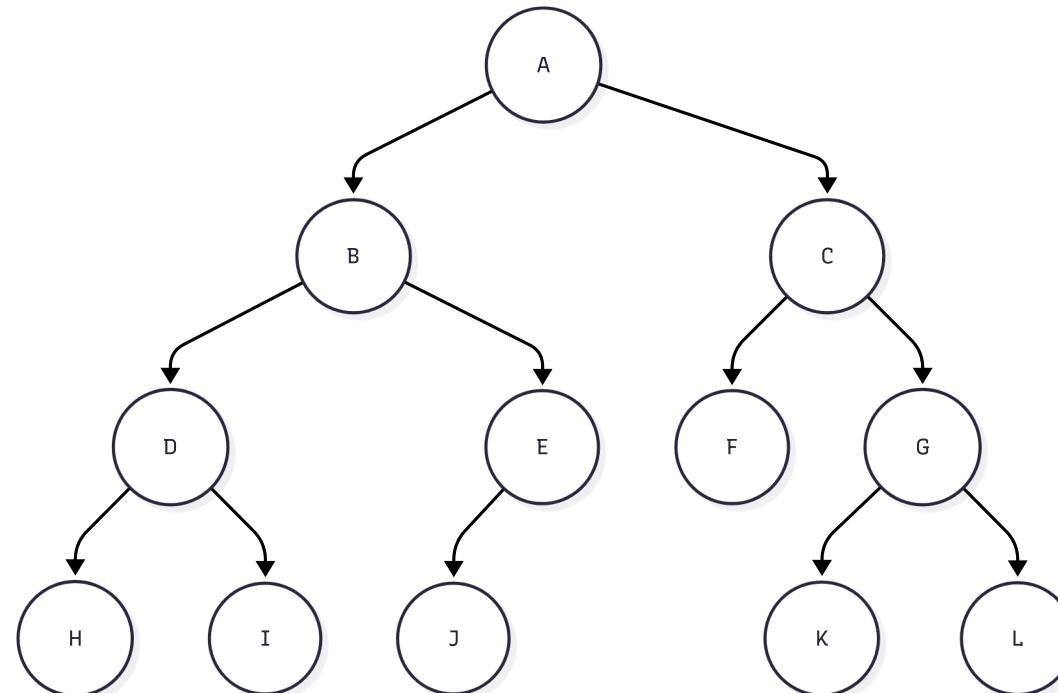
Árvore Binária Cheia

Cada nó possui zero ou dois filhos (**nunca exatamente um**). Na árvore ao lado, nenhum nó possui somente um filho. Portanto, ela é cheia.



Árvore Binária Completa

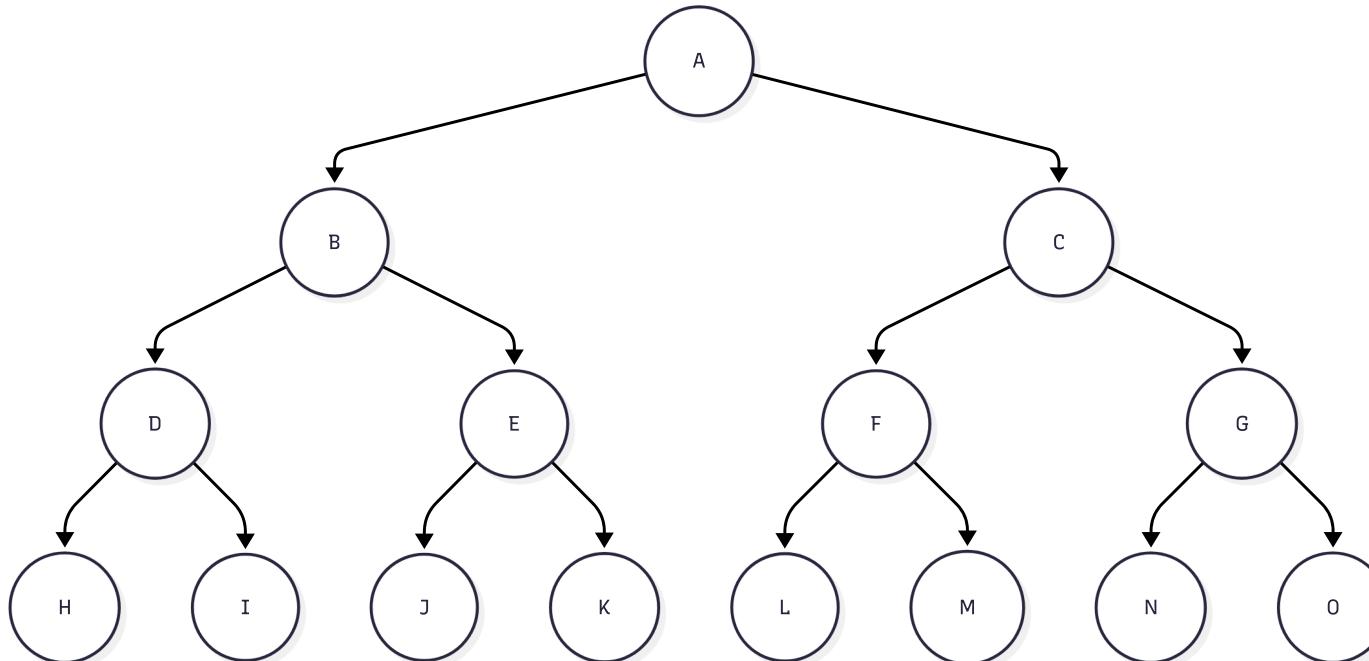
Neste caso, todos os níveis, acima das folhas do último nível, estão preenchidos integralmente.



A árvore acima possui 3 níveis, onde os níveis 0, 1 e 2 estão completos.

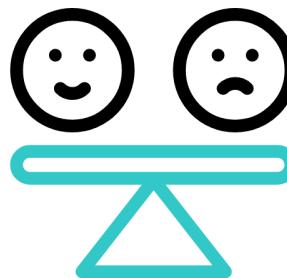
Árvore Binária Perfeita

É uma árvore binária em todos os níveis estão completamente preenchidos.



Todos os nós internos têm dois filhos e todas as folhas estão no mesmo nível.

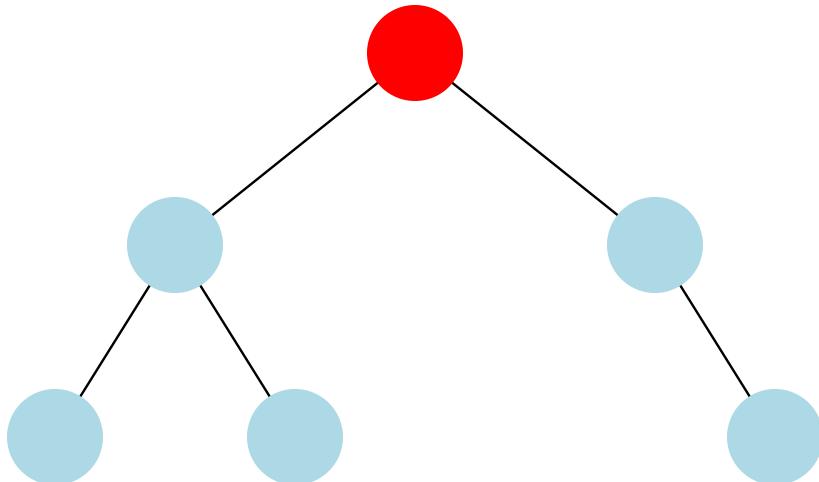
Observação



Mais a frente, estudaremos o balanceamento das árvores. Por hora, vale destacar a **árvore degenerada** que é aquela em que cada nó tem no máximo um filho. Isso faz com que a árvore se assemelhe a uma **lista encadeada**.

Questionamento

Um pergunta importante seria: **como podemos visitar todos os nós, sem exceção, de uma árvore somente uma vez?** Para isso, devemos estudar os percursos que podemos fazer em uma árvore.



Percursos

Definição

Percorso

Um **percurso**, ou **travessia**, diz respeito ao modo de como percorremos uma árvore. Percorrer uma árvore binária envolve **visitar todos os seus nós** em uma **ordem específica** para realizar várias operações, como pesquisar, classificar ou modificar a árvore.

As estratégias de travessia são amplamente classificadas em:

- Percurso em Largura
- Percurso em Profundidade

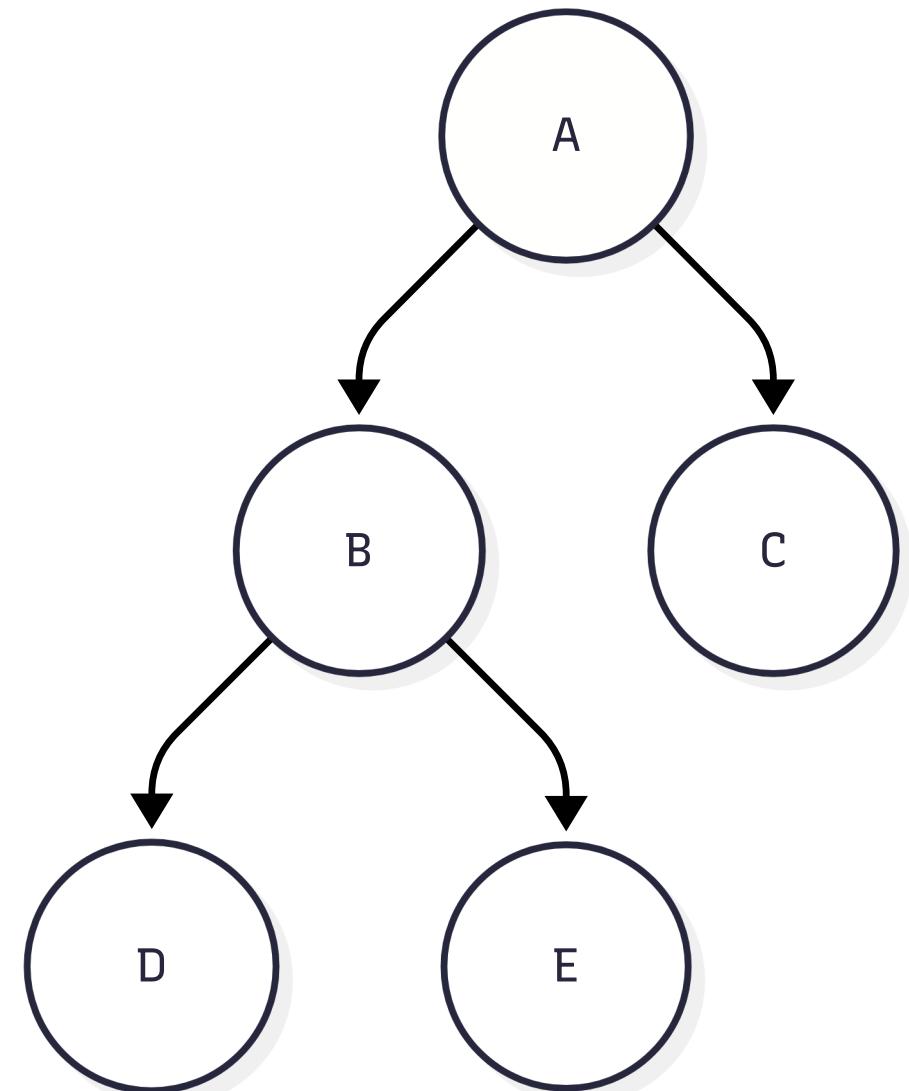
Percorso em Largura

O **percurso em largura** visita os nós em cada nível, da esquerda para a direita, antes de prosseguir para o próximo. Essa abordagem é ideal quando todos os nós em um nível precisam ser processados antes de passar para o próximo.

Exemplo

Um percurso em largura resultará na sequência A, B, C, D e E.

- No nível 0, passamos por A;
- No nível 1, passamos por B e C;
- No nível 2, passamos por D e E;



Percorso em Profundidade

O **percurso em profundidade** explora o mais profundamente possível em cada ramo antes de retroceder. Esta categoria inclui os subtipos:

- **pré-ordem**: nó, esquerda, direita
- **ordem**: esquerda, nó, direita
- **pós-ordem**: esquerda, direita, nó

Pré-ordem

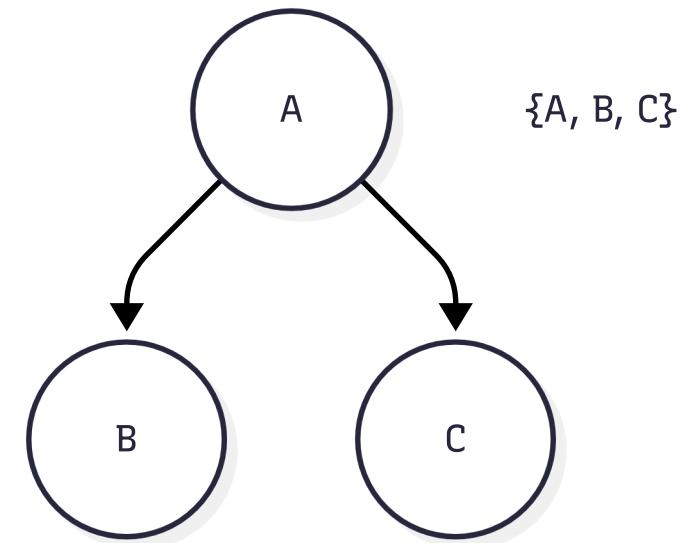
Pré-Ordem

Cada nó é processado antes de seus filhos. Segue a seguinte ordem de visita: raiz, subárvore esquerda, subárvore direita.

(raiz → esquerda → direita)

Exemplo

Processamos a raiz (**A**) e vamos para a subárvore esquerda. Processamos a nova raiz (**B**). Não temos subárvores de **B**, então voltamos e analisamos a subárvore direita de **A**. Processamos a nova raiz (**C**).



Ordem

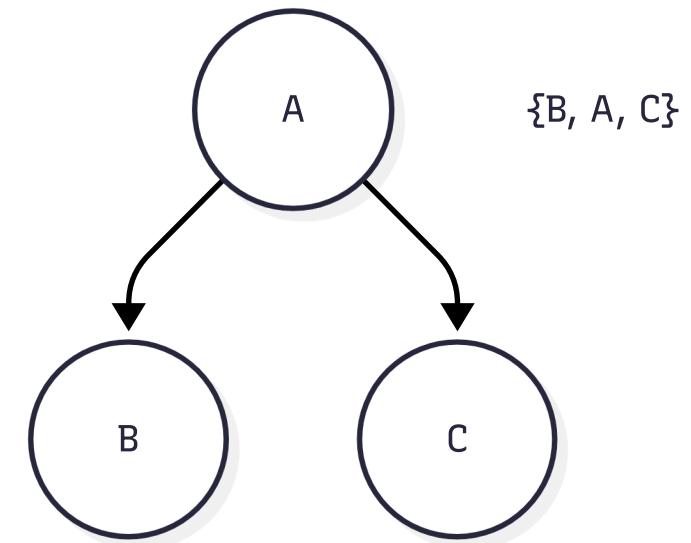
Ordem

Processa a subárvore esquerda, o nó atual e, em seguida, a subárvore direita.

esquerda → raiz → direita

Exemplo

Vamos para a subárvore esquerda. Processamos a nova raiz (**B**). Voltamos para a raiz e processamos (**A**). Vamos para a subárvore direita. Processamos a nova raiz (**C**).



Pós-ordem

Pós-Ordem

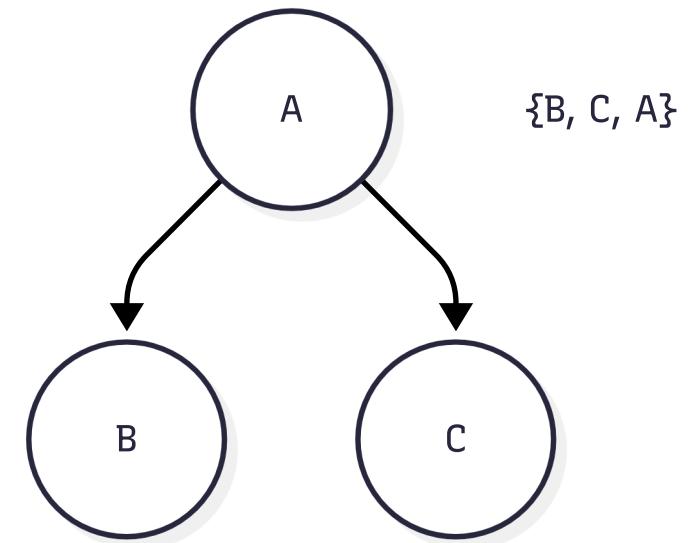
Acessa o nó atual após suas subárvore. Segue a seguinte ordem de visita: subárvore esquerda, subárvore direita e raiz.

esquerda → direita → raiz

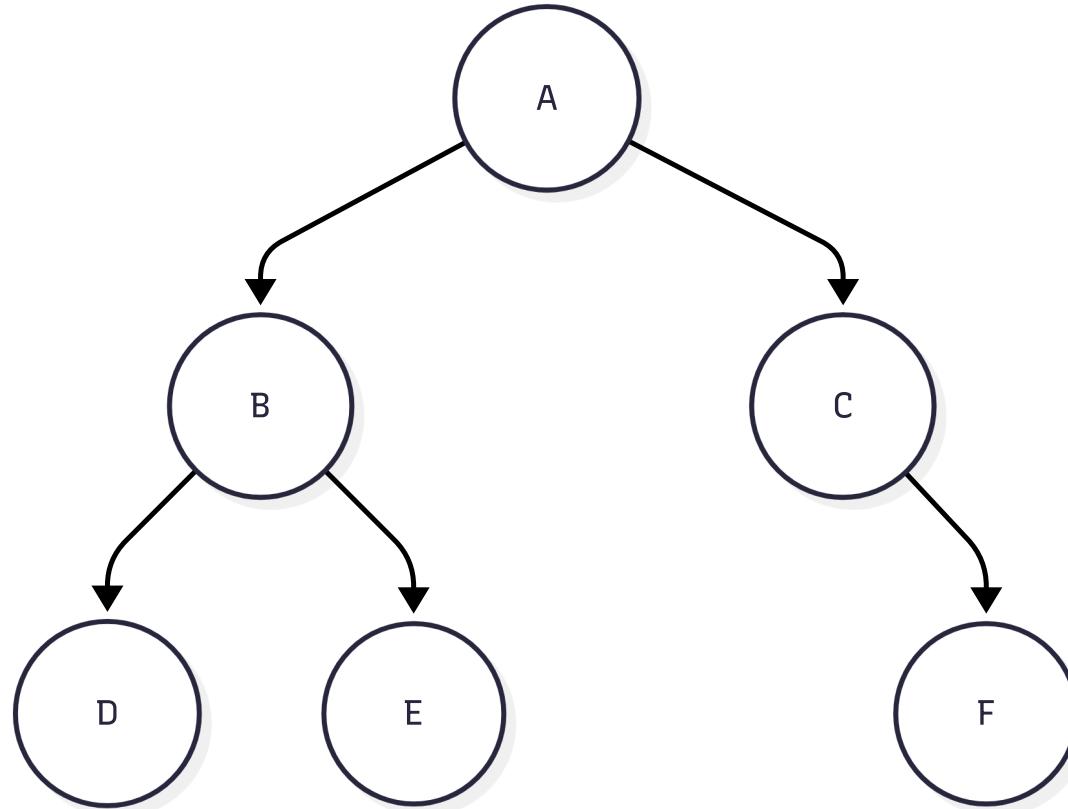
Exemplo

Vamos para a subárvore esquerda. Processamos a nova raiz (**B**). Vamos para a subárvore direita.

Processamos a nova raiz (**C**). Voltamos para a raiz e processamos (**A**).

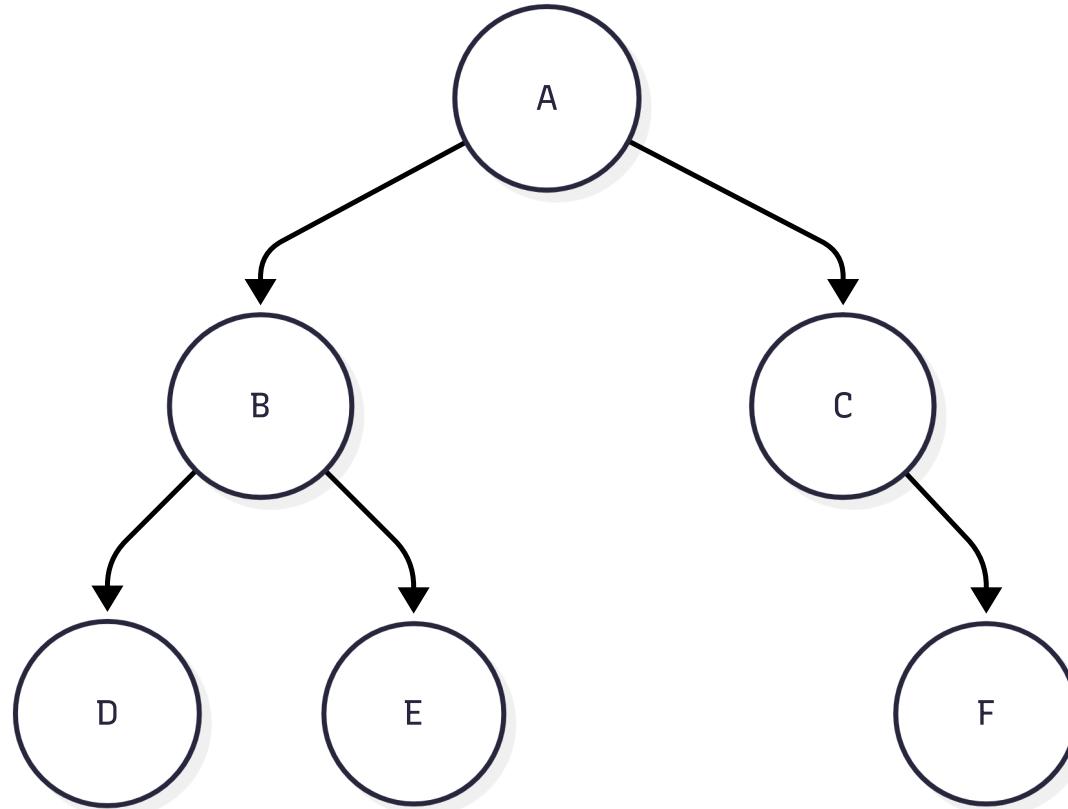


Fixação



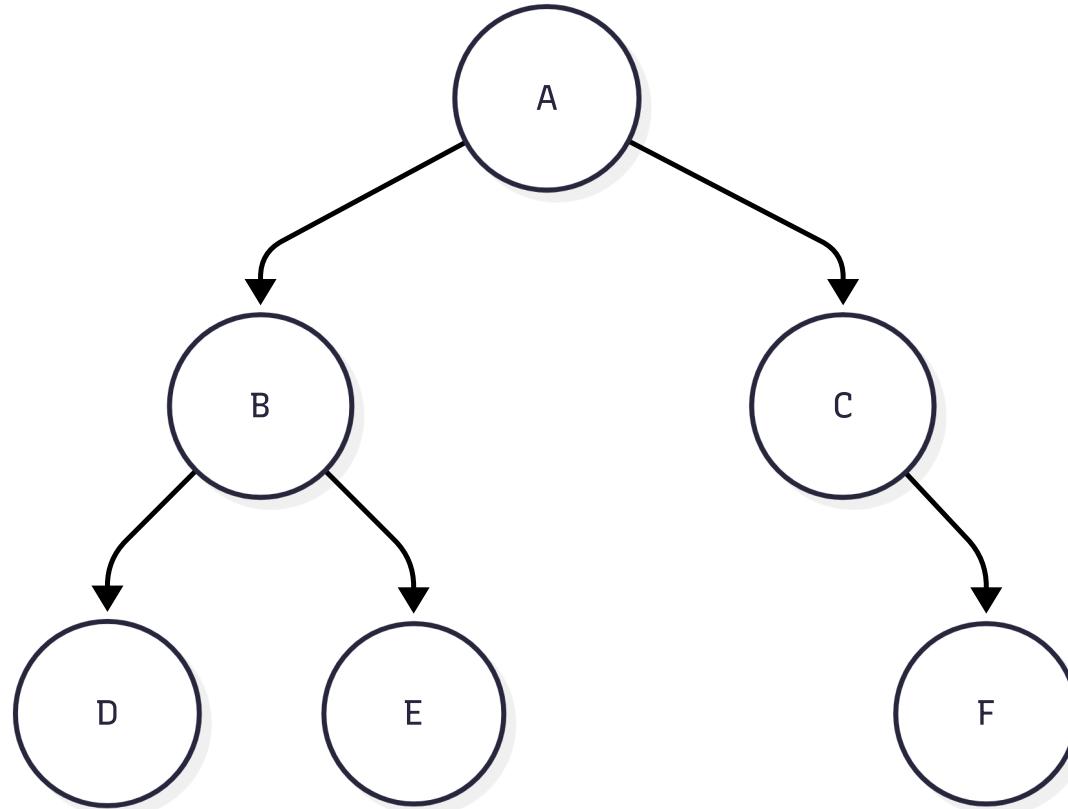
Largura: A, B, C, D, E, F

Fixação



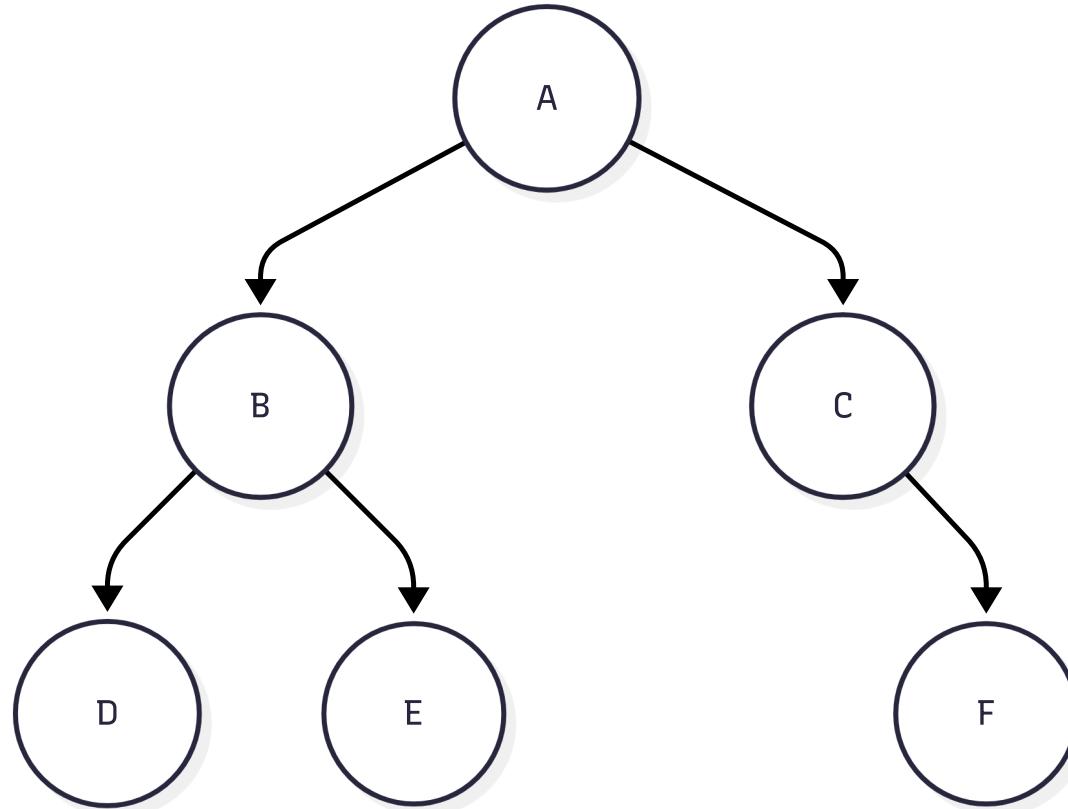
Pré-ordem (Nó, Esquerda, Direta): A, B, D, E, C, F

Fixação



Ordem (Esquerda, Nô, Direita): D, B, E, A, C, F

Fixação



Pós-ordem (Esquerda, Direta, Nó): D, E, B, F, C, A

Digreção

Parece complicado, mas com bastante prática e um pouco de artifício visual podemos estabelecer um método simples para determinar essas sequências.

Em sala de aula, estabelecemos um método fácil e rápido de encontrar a travessia, além de estudarmos o que acontece na memória *stack* quando a função é chamada recursivamente.

Agora, vamos partir para a implementação.

Implementação

Estrutura Nó

Para criarmos uma árvore binária em C, devemos criar uma nova estrutura. Tal estrutura deve conter o valor da chave e dois ponteiros para indicar os filhos do lado esquerdo e direito.



```
1 typedef struct node {
2     int data;
3     struct node *left;
4     struct node *right;
5 } Node;
```

Criando um Nó



```
● ● ●

1 Node* createNode(int key) {
2     Node* newNode = (Node*)malloc(sizeof(Node));
3
4     newNode->key = key;
5     newNode->left = newNode->right = NULL;
6
7     return newNode;
8 }
```

→ A função `createNode` recebe uma chave inteira e retorna um ponteiro para um objeto do tipo `Node`.

Alocando Memória



```
1 Node* createNode(int key) {  
2     Node* newNode = (Node*)malloc(sizeof(Node));  
3  
4     newNode->key = key;  
5     newNode->left = newNode->right = NULL;  
6  
7     return newNode;  
8 }
```

→ Na segunda linha, estamos alocando memória para o novo nó. Para isso, `sizeof(Node)` calcula o tamanho em bytes da estrutura `Node`. A função `malloc` retorna um ponteiro genérico, por isso fazemos um *casting* para `Node*`.

Inicialização



```
● ● ●
1 Node* createNode(int key) {
2     Node* newNode = (Node*)malloc(sizeof(Node));
3
4     newNode->key = key;
5     newNode->left = newNode->right = NULL;
6
7     return newNode;
8 }
```

→ Em seguida, preenchemos os campos do nó `newNode` : a chave e os filhos. No caso, o novo nó não possui filhos, então recebem o valor `NULL` .

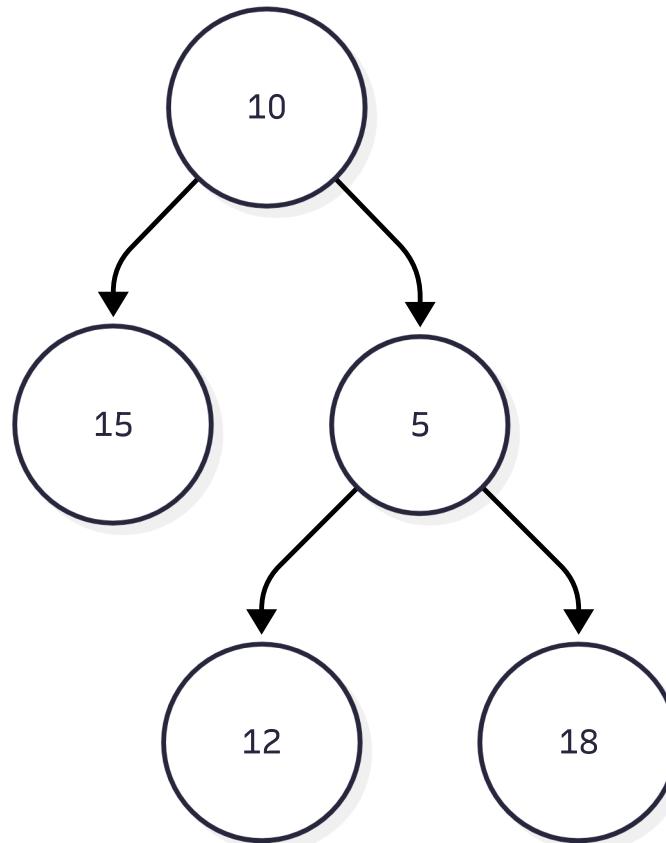
Criando uma árvore

Na função principal (`main`), podemos fazer a inserção da seguinte forma:

```
1 Node* root = createNode(10);
2     root->left = createNode(15);
3     root->right = createNode(5);
4     root->right->left = createNode(12);
5     root->right->right = createNode(18);
```

Resultado

O código anterior representaria a seguinte árvore:



Percursos

Os percusos são implementados recursivamente da seguinte forma:

- **Pré-ordem**: visita nó, processa esquerda, processa direita;
- **Ordem**: processa esquerda, visita nó, processa direita;
- **Pos-ordem**: processa esquerda, processa direita, visita nó;

Percursos



```
1 void preorder(Node* root) {  
2     if (root != NULL) {  
3         printf("%d ", root→key);  
4         preorder(root→left);  
5         preorder(root→right);  
6     }  
7 }
```



```
1 void inorder(Node* root) {  
2     if (root != NULL) {  
3         inorder(root→left);  
4         printf("%d ", root→key);  
5         inorder(root→right);  
6     }  
7 }
```



```
1 void posorder(Node* root) {  
2     if (root != NULL) {  
3         posorder(root→left);  
4         posorder(root→right);  
5         printf("%d ", root→key);  
6     }  
7 }
```

Percursos

Na pasta `code`, você pode encontrar a implementação no arquivo `arvorebinaria.c`. Desse modo, você pode criar árvores diversas e verificar as travessias implementadas.

No nosso exemplo anterior, o programa deve gerar a saída:

Pre-Ordem: `10 15 5 12 18`

Ordem: `15 10 12 5 18`

Pos-Ordem: `15 12 18 5 10`

Revisão e Considerações

Resumo

Nesta seção, estudamos a estrutura de dados árvore binária.

- Definimos essa estrutura;
- Classificamos as árvores binárias;
 - Cheia, completa e perfeita;
- Realizamos percursos diversos;
 - Largura e profundidade;

Quando ao percurso em profundidade, abordamos as travessias em pré-ordem, ordem e pós-ordem.

Considerações

Agora que estamos familiarizados com os conceitos fundamentais da estrutura de dados árvore, iremos estudar um tipo especial de árvore binária chamada **árvore binária de busca**.

Procure dominar os percursos realizando os exercícios propostos deste capítulo.



Obrigado



Prof. Dr. Bruno Xavier

Centro Multidisciplinar de Pau dos Ferros
Departamento de Engenharias e Tecnologia
Algoritmos e Estruturas de Dados 2

2025.2