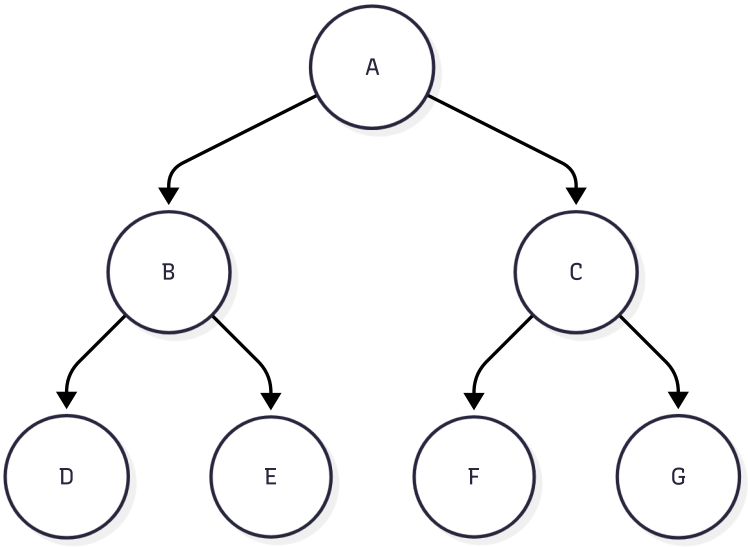
# Capítulo 2

# 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.

---




## 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**

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

**Exemplo**

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

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

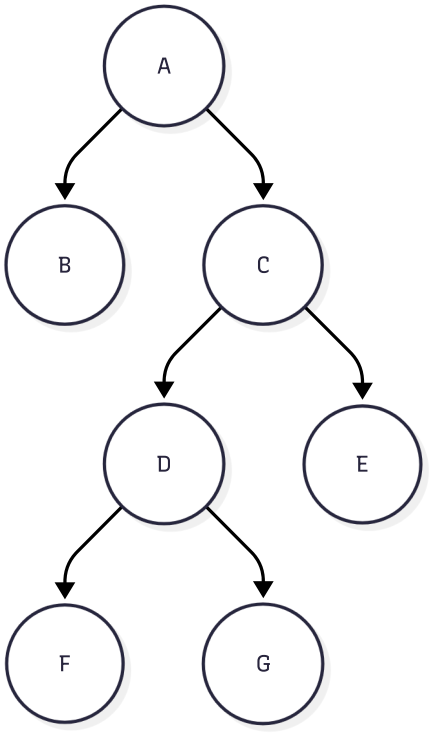
Além disso, tal árvore possuirá no máximo $\lceil 7/2 \rceil$ nós terminais, ou seja, $4$ folhas.

# Classificação das Árvore Binárias

Uma árvore binária pode ser classificada em cheia, completa e perfeita.

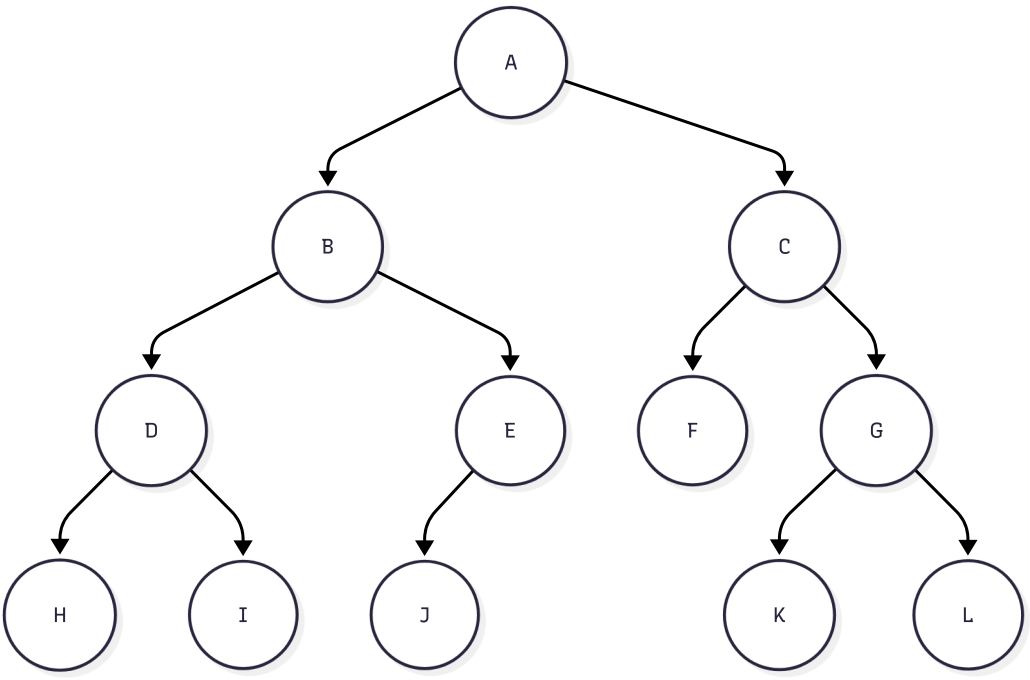
## Árvore Binária Cheia

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




## Árvore Binária Completa

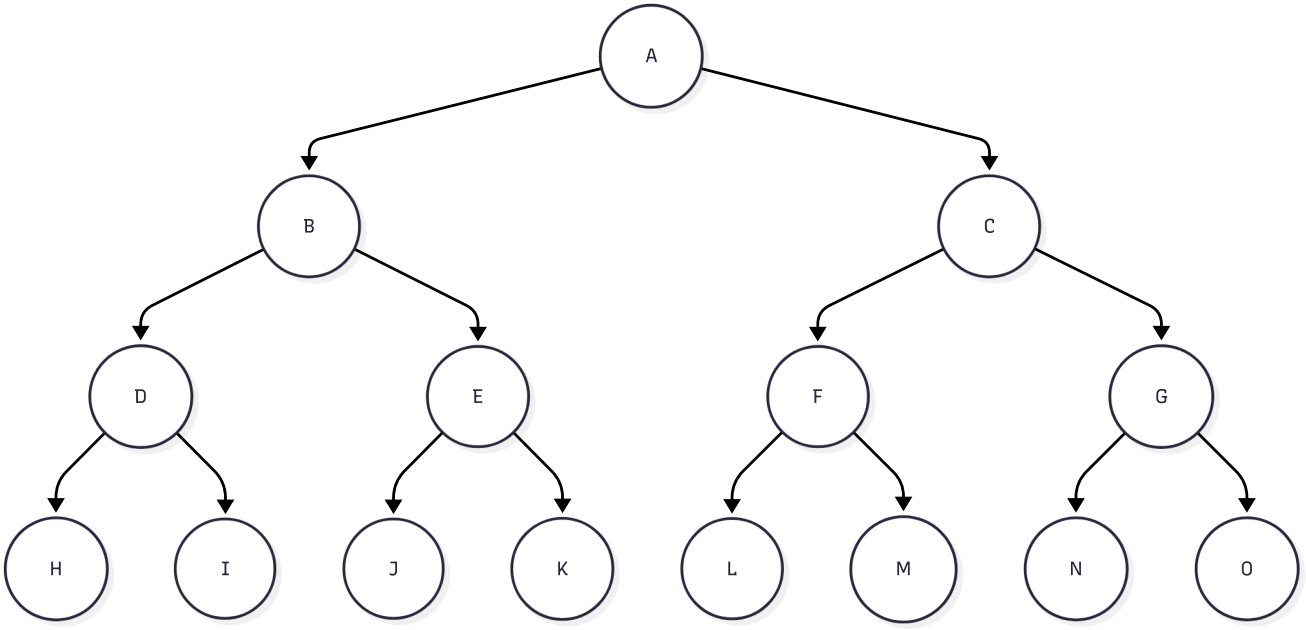
Todos os níveis, exceto talvez o último, estão completamente preenchidos. Este tipo de árvore é caracterizada por possuir todos os níveis, acima das folhas do último nível, preenchidos integralmente.




A árvore acima possui 3 níveis, onde os níveis 0, 1 e 2 estão completos, ou seja, todos os nós possuem 2 filhos.

## Árvore Binária Perfeita

É uma árvore binária cheia e todos os níveis estão completamente preenchidos




Note que todos os nós internos têm dois filhos e todas as folhas estão no mesmo nível.

Além disso, a árvore está bem equilibrada, com os ramos esquerdo e direito balanceados. Em contraste a uma árvore equilibrada há a **árvore degenerada**

que é aquela em que cada nó tem no máximo um filho, exceto o nó folha. Isso faz com que a árvore se assemelhe a uma **lista encadeada**.

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

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.

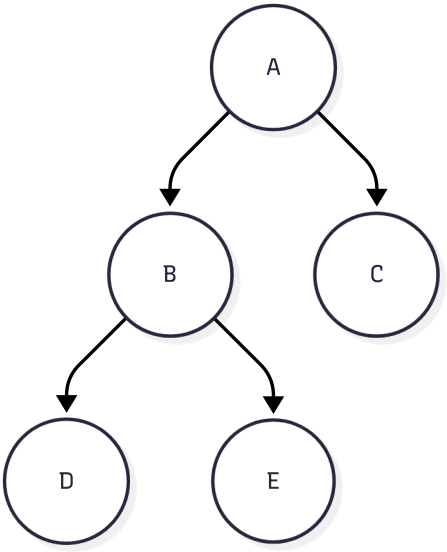
As estratégias de travessia são amplamente classificadas em duas categorias:

- Percurso em Largura
- Percurso em Profundidade

## Percurso 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 executada usando uma fila e é ideal quando todos os nós em um nível precisam ser processados antes de passar para o próximo.

Considere a árvore abaixo:



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;

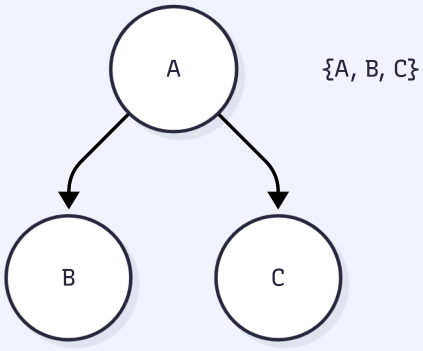
## Percurso em Profundidade

O **percurso em profundidade** explora o mais profundamente possível em cada ramo antes de retroceder. Esta categoria inclui os subtipos pré-ordem, ordem e pós-ordem.

### Pré-ordem

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

### Exemplo

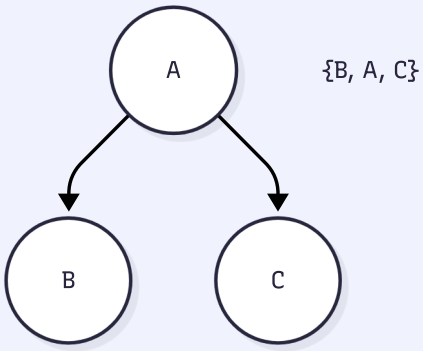



Processamos a raiz (A) e vamos para a subárvore esquerda. Processamos a nova raiz (B). Não temos subárvores de B, então voltamos e analisamos a subárvore direita de A. Processamos a nova raiz (C).

### Ordem

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

### Exemplo

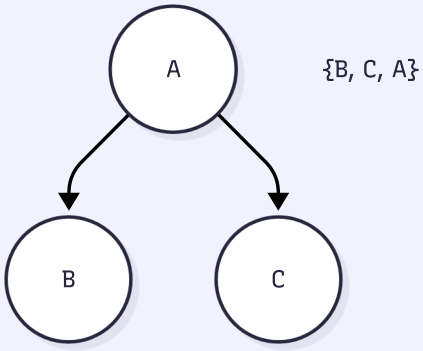



Vamos para a subárvore esquerda. Processamos a nova raiz (B). Voltamos para a raiz e processamos (A). Vamos para a subárvore direita. Processamos a nova raiz (C).

### Pós-ordem

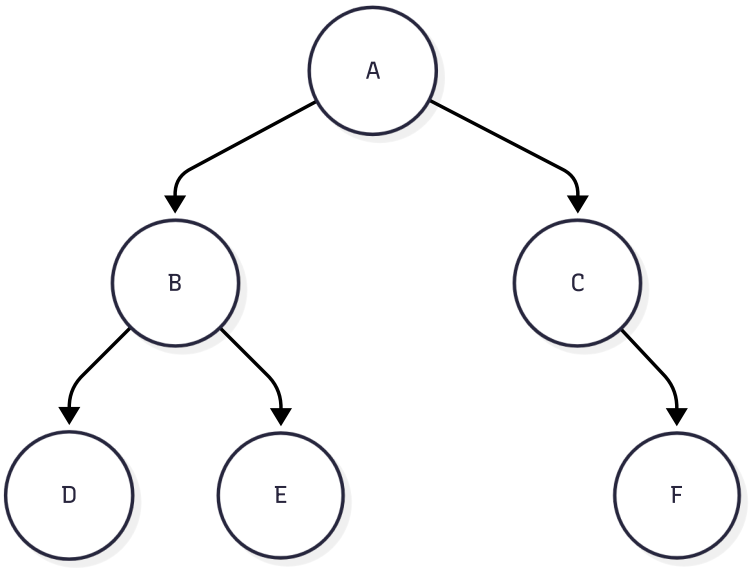
Acessa o nó atual após suas subárvores. Segue a seguinte ordem de visita: subárvore esquerda, subárvore direita e raiz.

### 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).

Outro exemplo a seguir:




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

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

# Implementação

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.

```c
typedef struct node {
    int data;
    struct node *left;
    struct node *right;
} Node;
```

Para criar um novo nó, podemos definir a seguinte função.

```c
Node* createNode(int key) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->key = key;
    newNode->left = newNode->right = NULL;
```

```
    return newNode;
}
```

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

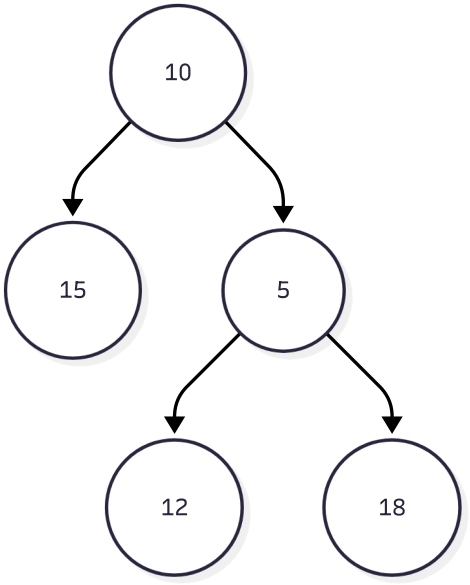
Na segunda linha, estamos alocando memória para o novo nó. Para isso, `sizeof(Node)` calcula o tamanho em bytes da estrutura `Node`. A função `malloc` retorna um ponteiro genérico, por isso fazemos um *casting* para `Node*`.

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`.

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

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

Isso geraria a seguinte árvore:



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.