VOLTAR

# Árvores

Apresentar o conceito sobre árvores e mostrar alguns exemplos da utilização dessa estrutura de armazenamento de dados.

NESTE TÓPICO

Marcar tópico

## Introdução

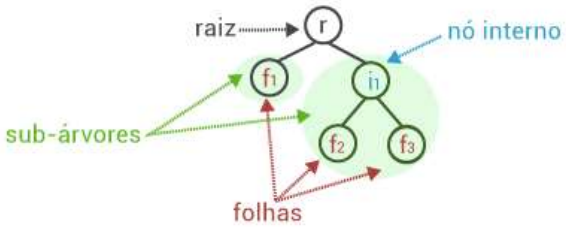
As Listas Simplesmente e Duplamente Encadeadas, Fila e Pilha podem ser consideradas listas lineares, sejam elas estáticas ou dinâmicas. Embora tais listas apresentem vantagens quanto ao uso, à manipulação e à alocação, ainda possuem problemas:

- Lista encadeada
  - Eficiente para inserção e remoção dinâmica de elementos, mas ineficiente para busca;
- Lista seqüencial (ordenada)
  - Eficiente para busca, mas ineficiente para inserção e remoção de elementos.

Em busca de contornar essas desvantagens, foi proposto o conceito de Árvores, que apresenta solução eficiente para inserção, remoção e busca. Vale destacar que as árvores possuem uma representação não linear.

As árvores são estruturas de dados adequadas para a representação de hierarquias. A forma mais natural para definirmos uma estrutura de árvore é usando recursividade. Uma árvore é composta por um conjunto de nós. Existe um nó r, denominado raiz, que contém zero ou mais sub-árvores, cujas raízes são ligadas diretamente a r.

Esses nós raízes das sub-árvores são ditos filhos do nó pai, r. Nós com filhos são comumente chamados de nós internos e nós que não têm filhos são chamados de folhas, ou nós externos. É tradicional desenhar as árvores com a raiz para cima e folhas para baixo, ao contrário do que seria de se esperar. A figura a seguir exemplifica a estrutura de uma árvore.




Estrutura de uma árvore.

O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes. Serão estudados dois tipos de árvores. Primeiro, examinaremos as árvores binárias, onde cada nó tem, no máximo, dois filhos. Depois examinaremos as chamadas árvores genéricas, onde o número de filhos é indefinido. Estruturas recursivas serão usadas como base para o estudo e a implementação das operações com árvores.

## Definição

Árvore T – conjunto finito de elementos, denominados nós ou vértices, tais que:

- Se T = Ø, a árvore é dita vazia;
- Caso contrário:

i. T contém um nó especial, denominado raiz;

ii. os demais nós ou constituem um único conjunto vazio, ou são divididos em $m^{3}1$ conjuntos disjuntos não vazios ($T_1$,$T_2$,...,$T_n$), que são, por sua vez, cada qual uma árvore;

- $T_1$,$T_2$,...,$T_n$ são chamadas sub-árvores de T;
- Um nó sem sub-árvores é denominado nó-folha, ou simplesmente, folha.

Se um nó T é a raiz de uma árvore e um nó $T_1$ é raiz de uma sub-árvore de T, então T é o PAI de $T_1$ e $T_1$ é o FILHO de T.

## Conceitos Iniciais

1) NÍVEL: o nível de um nó T é definido como:

- O nível de um nó raiz é 1;
- O nível de um nó não raiz é dado por (nível de seu nó PAI + 1).

2) PROFUNDIDADE: a profundidade de uma árvore é dada pelo nível máximo de qualquer nó folha na árvore, ou analogamente, pelo número de nós do maior percurso do nó raiz até qualquer nó folha.

3) GRAU: o grau de um nó T de uma árvore é igual ao número de filhos do nó T;

4) GRAU DA ÁRVORE: o grau de uma árvore T é o grau máximo entre os graus de todos os seus nós;

5) ALTURA: A altura de um nó v é o número de nós no maior caminho de v até um de seus descendentes. Os nós folha sempre têm altura igual a 1;

6) ALTURA DA ÁRVORE: a altura de uma árvore T é dada pela altura máxima de seus nós.

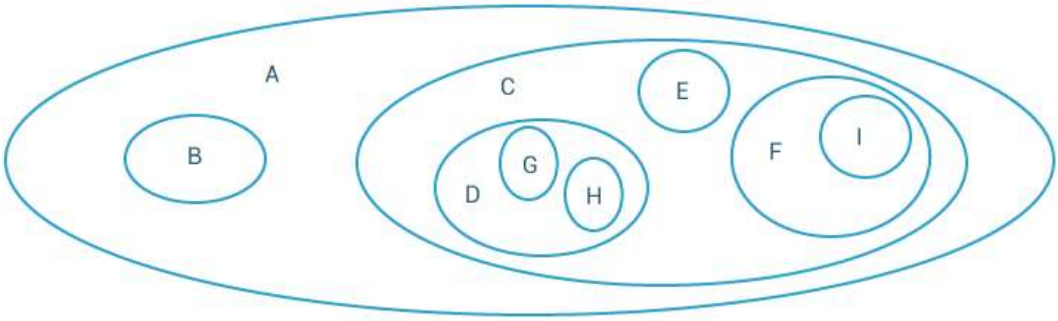
- Denota-se a altura de uma árvore com raiz dada pelo nó T por h(T), e a altura de uma sub-árvore com raiz $T_1$ por $h(T_1)$.

- Outras formas de representação:

- Representação por parênteses aninhados

( A (B) ( C (D (G) (H)) (E) (F (I)) ) )

- Representação por Diagramas de Venn (Figura abaixo).




Árvore representada por um diagrama de Venn

Os conceitos iniciais sobre as Árvores são ilustrados na Figura a seguir.

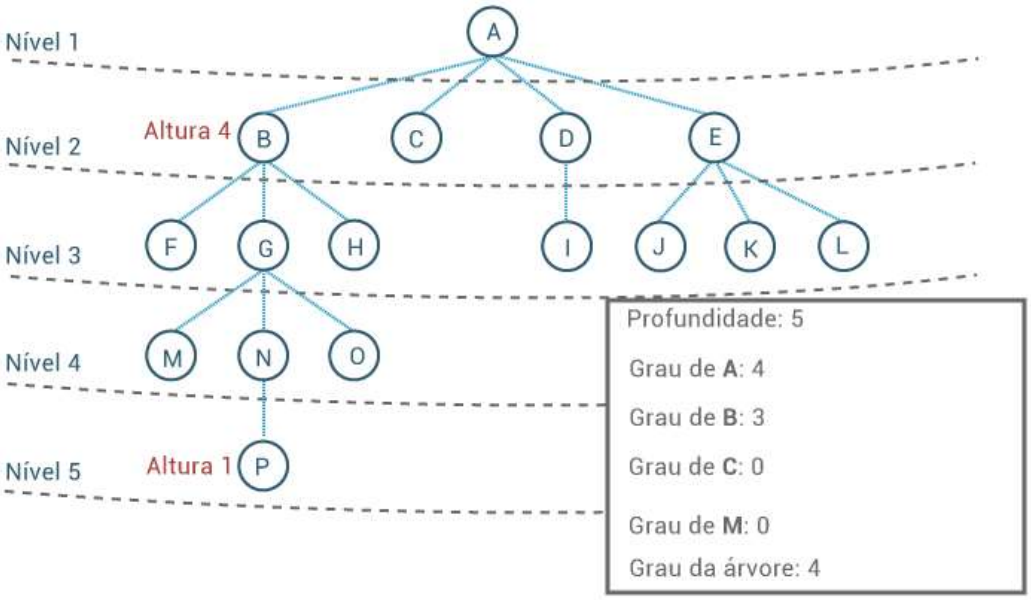



Ilustração dos conceitos iniciais sobre Árvores

Além disso, mediante a representação hierárquica, pode-se definir que:

O nó A é ANCESTRAL dos nós B, C, D, E ... P.

- Um nó X é um ANCESTRAL do nó Y (e Y é DESCENDENTE de X) se X for o PAI de Y ou então se X for o PAI de algum ANCESTRAL de Y;

O nó B é irmão de C, D e E.

- Dois nós são IRMÃOS se forem filhos do mesmo pai.

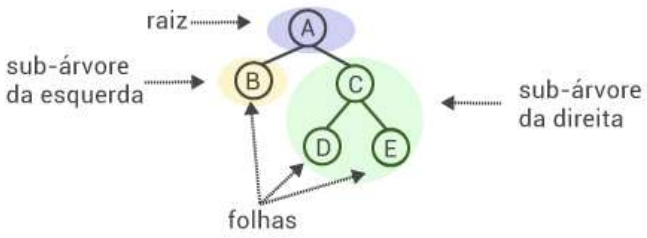
Por fim, uma FLORESTA é o conjunto de Árvores disjuntas.

# Árvores Binárias

Uma árvore binária é um conjunto finito de elementos que ou é vazio ou é dividido em três subconjuntos disjuntos:

- A raiz da árvore;
- Uma árvore binária chamada de sub-árvore da esquerda;
- Uma árvore binária chamada de sub-árvore da direita.

A Figura abaixo ilustra o conceito de Árvores Binárias.




Representação de uma árvore binária

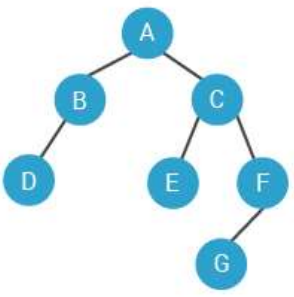
Uma árvore binária possui algumas propriedades:

- O número máximo de nós no nível $i$ de uma árvore binária é $2^{i-1}$.

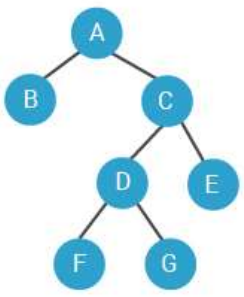
- O número máximo de nós em uma árvore binária de altura $k$ é $2^{k}-1$.

# Árvores Estritamente Binárias

Cada nó que não for folha em uma árvore binária tem sub-árvores esquerda e direita não vazias. Uma árvore estritamente binária com $n$ folhas tem $2n-1$ nós. Nós interiores (não folhas) possuem sempre 2 filhos. A Figura abaixo mostra a diferença de uma árvore binária comum e uma árvore estritamente binária.




Árvore Binária



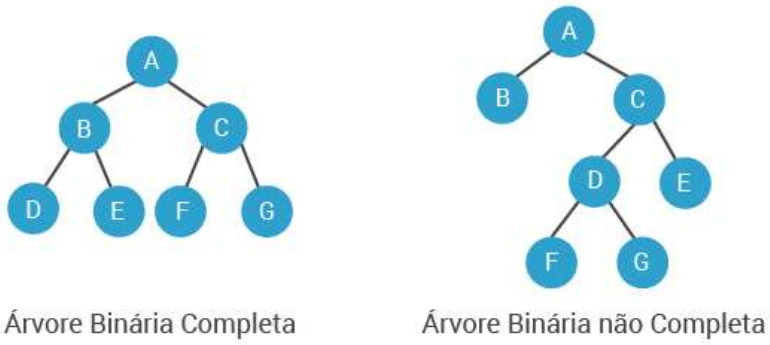

Árvore Estritamete Binária

Diferença de uma árvore binária comum e uma árvore estritamente binária

## Árvore Binária Completa

Uma Árvore Binária Completa de Profundidade *k* é uma *árvore estritamente binária* nas quais todas as folhas estão no nível *k*.

Se *k* é a profundidade da árvore, o número total de nós é $2^k-1$. O número total de folhas é $2^{k-1}$. A Figura abaixo mostra a diferença de uma árvore binária completa e uma árvore binária não completa.



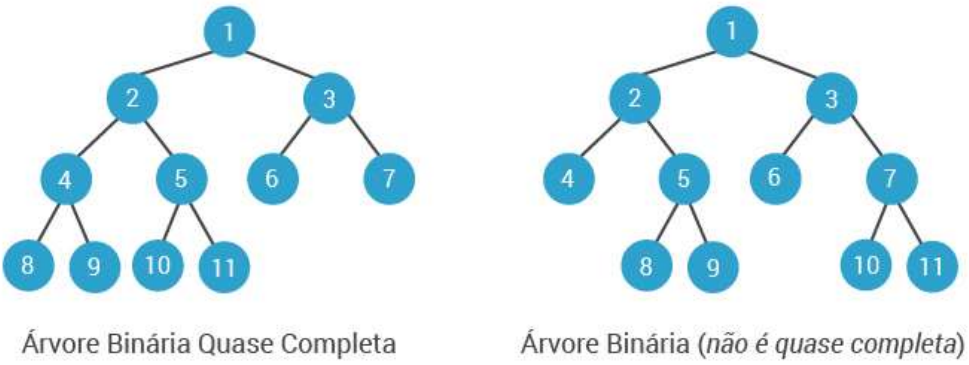

Diferença de uma árvore binária completa e uma árvore binária não completa

## Árvore Binária Quase Completa

É uma Árvore Binária com profundidade *k* onde:

- Cada nó folha na árvore está no nível *k* ou no nível *k-1* (até o nível *k-1* ela é completa);
- Para qualquer nó *n* da árvore com um descendente direito no nível *k*, também deve existir o descendente esquerdo correspondente no nível *k*.
- Uma árvore binária quase completa pode ter seus nós numerados começando da raiz, de cima para baixo e da esquerda para a direita sem que haja a ausência de nós.

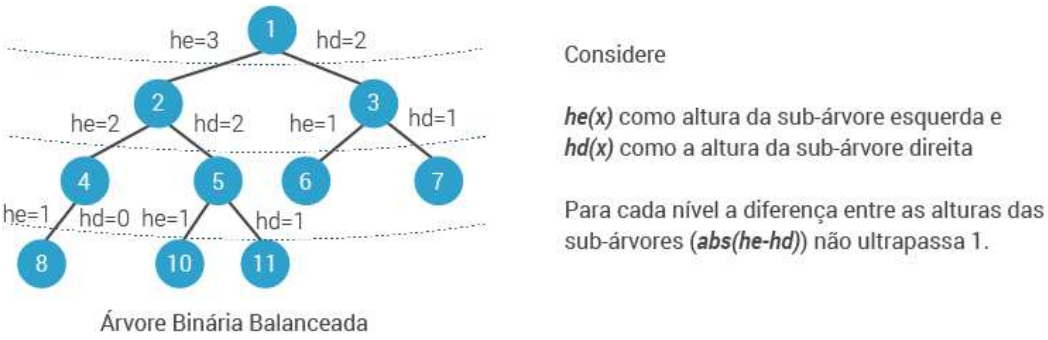
A Figura abaixo mostra a diferença de uma árvore binária quase completa e uma árvore binária que não é quase completa.




Diferença de uma árvore binária quase completa e uma árvore binária que não é quase completa

## Árvores Balanceadas

Uma árvore balanceada é aquela onde para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1. A Figura abaixo mostra o exemplo de uma Árvore Binária Balanceada.

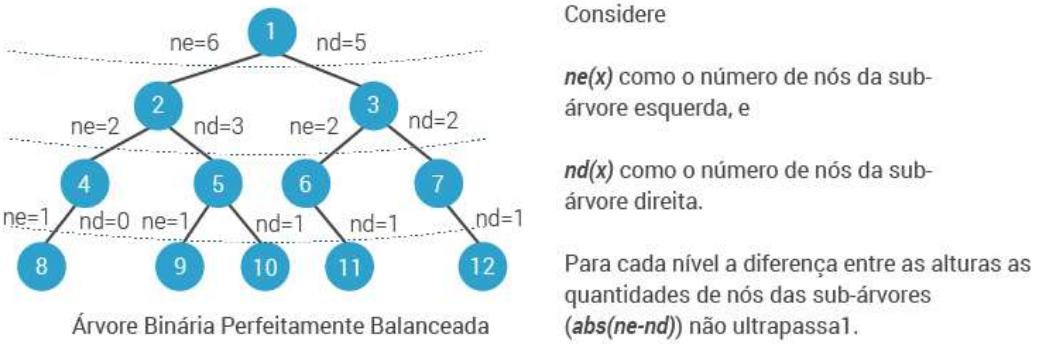



Representação de uma árvore binária balanceada

## Árvore Binária Perfeitamente Balanceada

O número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1.

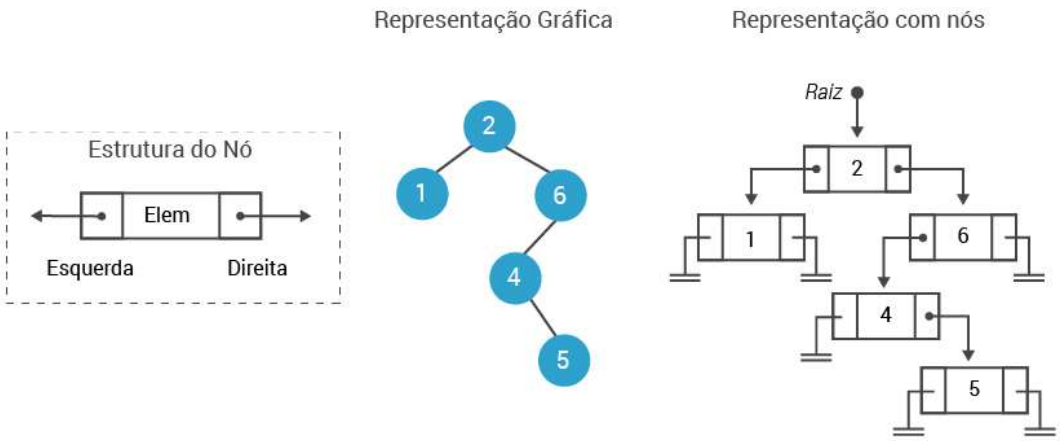
A Figura abaixo mostra o exemplo de uma Árvore Binária Perfeitamente Balanceada.




Representação de uma árvore perfeitamente balanceada

## Implementação da Árvore Binária

A Figura abaixo ilustra a representação gráfica de um nó para qualquer árvore binária.




Representação gráfica de um nó para a árvore binária

A seguir é mostrada a codificação de um programa para a criação de uma árvore binária.

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct no_arv
{
 int info;
 struct no_arv *esq;
 struct no_arv *dir;
};

typedef struct no_arv *tipo_arv;
tipo_arv raiz=NULL;
//*****************************************************************************

int arvoreVazia(){
   if (raiz==NULL)
       return(1);
   return(0);
}
//*****************************************************************************

tipo_arv criaRaiz(int x){
   tipo_arv novoNo= (tipo_arv) malloc(sizeof(no_arv));
   novoNo->info = x;
   novoNo->esq=NULL;
   novoNo->dir=NULL;
   raiz = novoNo;
   return (novoNo);
}
//*****************************************************************************
tipo_arv insereDir(tipo_arv *pai, int x){
 tipo_arv no_pai = *pai;
 if (!arvoreVazia()){
    if (no_pai->dir!=NULL){
        printf("\n No direito ja ocupado" );
        return NULL;
    }
    else{
         tipo_arv novoNo= (tipo_arv)malloc(sizeof(no_arv));
         novoNo->info = x;
         novoNo->esq=NULL;
         novoNo->dir=NULL;
         no_pai->dir = novoNo;
         return(novoNo);
     }
}
 return NULL;
}
tipo_arv insereEsq(tipo_arv *pai, int x)
{
 tipo_arv no_pai = *pai;
 if (!arvoreVazia()){
    if (no_pai->esq!=NULL){
        printf("\n No esquerdo ja ocupado" );
        return NULL;
    }
    else{
         tipo_arv novoNo= (tipo_arv)malloc(sizeof(no_arv));
         novoNo->info = x;
         novoNo->esq=NULL;
         novoNo->dir=NULL;
         no_pai->esq = novoNo;
         return(novoNo);
     }
}
 return NULL;
}
//*****************************************************************************
// principal
//*****************************************************************************
```

```
72.  int main()
73.  {
74.      tipo_arv No;
75.      raiz = criaRaiz(10);
76.      No = raiz;
77.      insereDir(&No,12);
78.      No = insereEsq(&No,5);
79.      insereEsq(&No,2);
80.      No = insereDir(&No,8);
81.      getch();
82.  }
```

## Percursos em Árvores Binárias

Uma vez que determinadas informações são armazenadas em uma estrutura como uma árvore, é preciso de alguma maneira acessar as informações lá contidas. Se fosse uma lista linear, bastaria percorrer essa lista do início ao fim um após o outro de forma seqüencial. Entretanto, uma árvore é uma estrutura não linear e dessa maneira, por onde iniciar o percurso? Onde encerrar? *É necessário ressaltar que em uma árvore binária não há nenhuma relação de ordem dentre os valores!*

Pode-se dizer que o objetivo, então, é percorrer uma Árvore Binária 'visitando' cada nó uma única vez. Um percurso gera uma seqüência linear de nós, e dessa forma é possível falar de nó predecessor ou sucessor de um nó, segundo um dado percurso. Em outras palavras, obter uma seqüência a partir de uma estrutura não linear!

Não existe um percurso único para árvores (binárias ou não): diferentes percursos podem ser realizados, dependendo da aplicação. A utilização é imprimir uma árvore, remover um item, buscar por um item, entre outras.

Há três percursos básicos para Árvores Binárias:

- pré-ordem (*Pre-order*) ou profundidade;
- em-ordem (*In-order*) ou ordem simétrica;
- pós-ordem (*Post-order*).

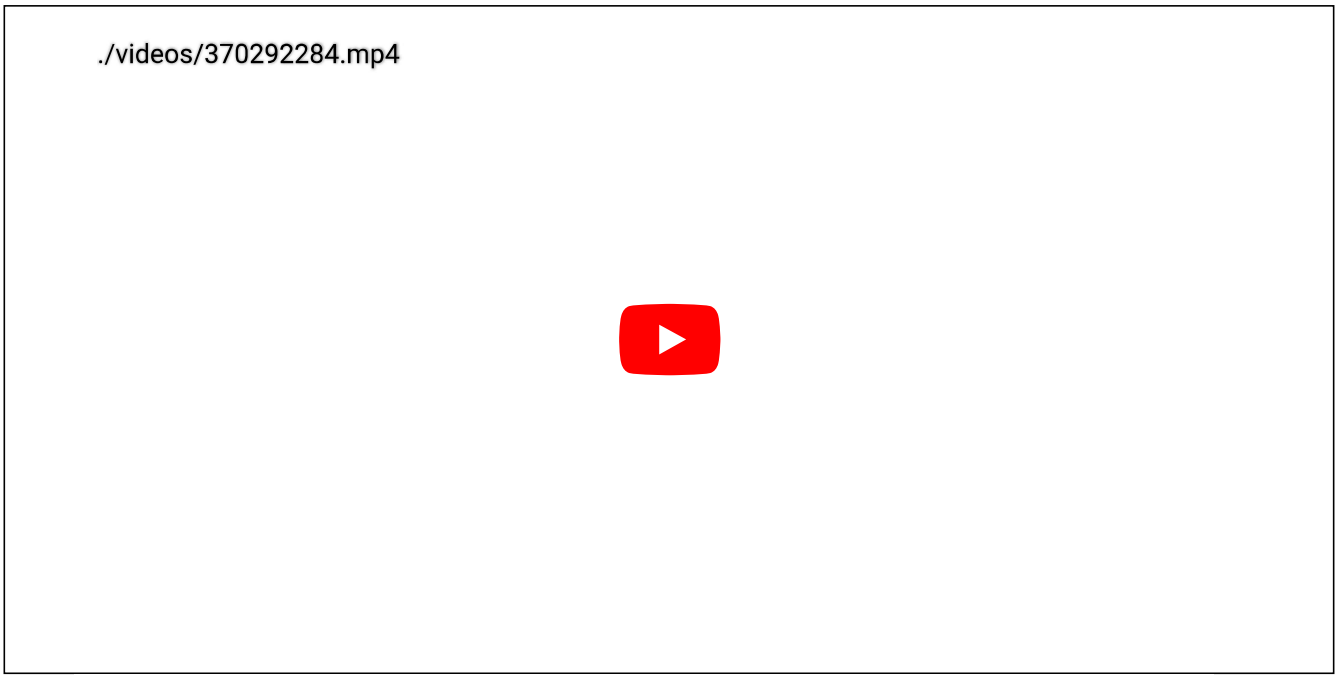
A diferença entre esses percursos é, basicamente, a ordem em que os nós são "visitados".

## Percurso Pré-Ordem ou Profundidade

O percurso pré-ordem consiste nos seguintes passos:

1. Mostra o valor do nó;
2. Visita o nó esquerdo;
3. Visita o nó direito;

A Figura abaixo ilustra o percurso pré-ordem em uma árvore binária.




Abaixo, segue a implementação desse percurso.

```
1.  void percursoPreOrdem(){
2.        preordem(raiz);
3.  }
4.  void preordem(tipo_arv r){
5.          if (r != null) {
6.                  printf ("%d ,", r->info);
7.                  preordem (r->esq);
8.                  preordem (r->dir);
9.          }
10. }
```

## Percurso Em-Ordem ou Ordem Simétrica

O percurso em-ordem consiste nos seguintes passos:

1. Visita o nó esquerdo;
2. Mostra o valor do nó;
3. Visita o nó direito;

A Figura abaixo ilustra o percurso em-ordem em uma árvore binária.

./videos/370366704.mp4

Abaixo, segue a implementação desse percurso.

```
1.  void percursoEmOrdem(){
2.        emordem(raiz);
3.  }
4.  void emordem(tipo_arv r){
5.          if (r != null) {
6.                  emordem (r->esq);
7.                     printf ("%d ,", r->info);
8.                  emordem (r->dir);
9.          }
10. }
```

## Percurso Pós-Ordem

O percurso pós-ordem consiste nos seguintes passos:

1. Visita o nó esquerdo;
2. Visita o nó direito;
3. Mostra o valor do nó.

A Figura abaixo ilustra o percurso pós-ordem em uma árvore binária.

./videos/370292392.mp4

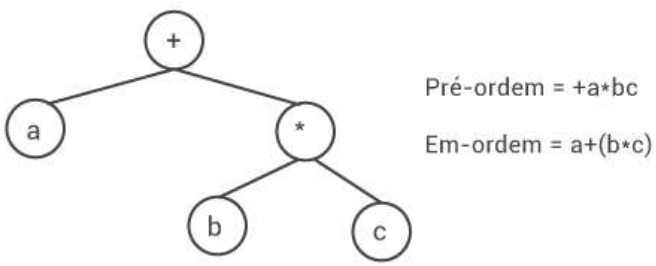
Abaixo, segue a implementação desse percurso.

```
1.  void percursoPosOrdem(){
2.        posordem(raiz);
3.  }
4.  void posordem(tipo_arv r){
5.          if (r != null) {
6.                  posordem (r->esq);
7.                     posordem (r->dir);
8.                     printf ("%d ,", r->info);
9.          }
10. }
```

Esses percursos são implementados recursivamente, pois permite um algoritmo simples, versátil e que não compromete o desempenho da máquina, pois em geral tais estruturas, quando balanceadas, se utilizam de uma pilha de recursão correspondente à altura da árvore. Em algoritmos iterativos é preciso o uso de uma pilha, ou então uma referência em cada nó ao nó Pai respectivo.

## Aplicação de Percursos para Expressões Aritméticas

Abaixo é mostrado um exemplo do uso de percursos em árvores binárias para expressões aritméticas.




Exemplo de percursos para expressões aritméticas

## Quiz

Exercício

Árvores

INICIAR

## Quiz

Exercício Final

Árvores

INICIAR

# Referências

MIZRAHI, V. V. *Treinamento em linguagem C*, São Paulo: Pearson, 2008.

SCHILDT, H. C – Completo e Total, São Paulo: Pearson, 2006.

Avalie este tópico

ANTERIOR
Recursividade

Índice

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site



Ajuda? (https://ava.uninove.br/seu/AVA/ajuda/ajuda.php?idCurso=)