Estrutura de Dados

Aula 13 - Árvores Binárias

Prof. MSc. Fausto Sampaio

fausto.sampaio.unifanor.edu.br

Centro Universitário UniFanor - Wyden

3 de dezembro de 2019

Sumário

Objetivos

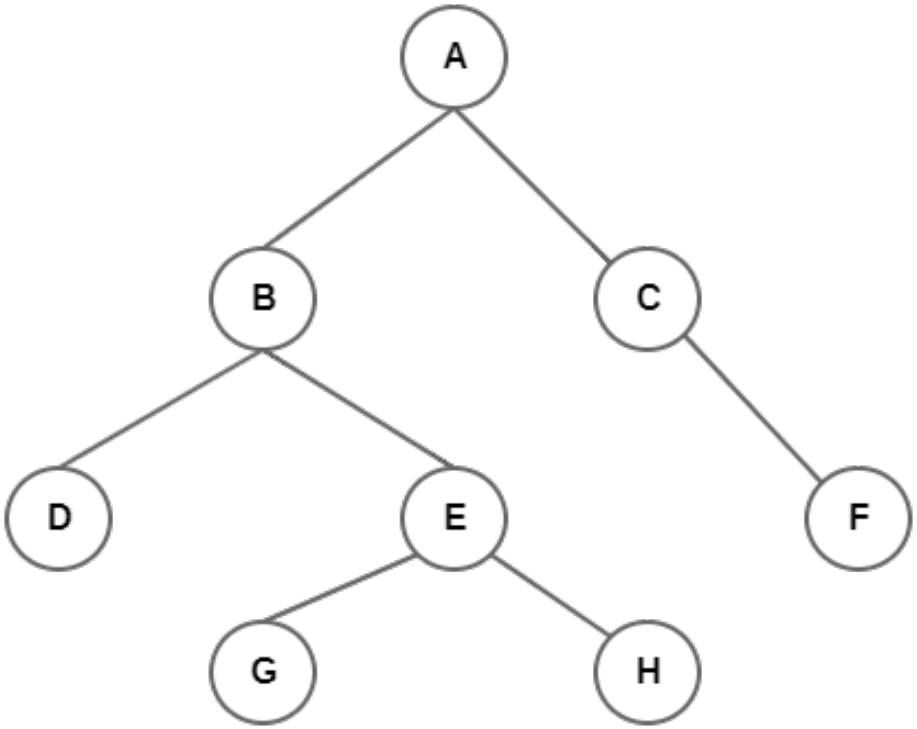
Objetivos

- Conceituar a estrutura de dados em Árvore Binária;
- Implementar uma estrutura em Árvore Binária e suas operações básicas.

Árvore Binária

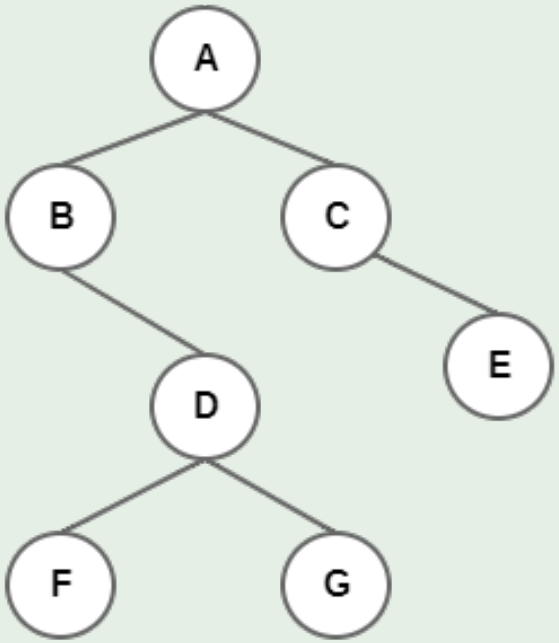
Definição

- **Árvore Binária**: é um tipo de árvore.
- Cada nó pode possuir duas sub-árvore:
  - sub-árvore esquerda;
  - sub-árvore direita;
- O grau de cada nó (número de filhos) pode ser 0, 1 ou 2;

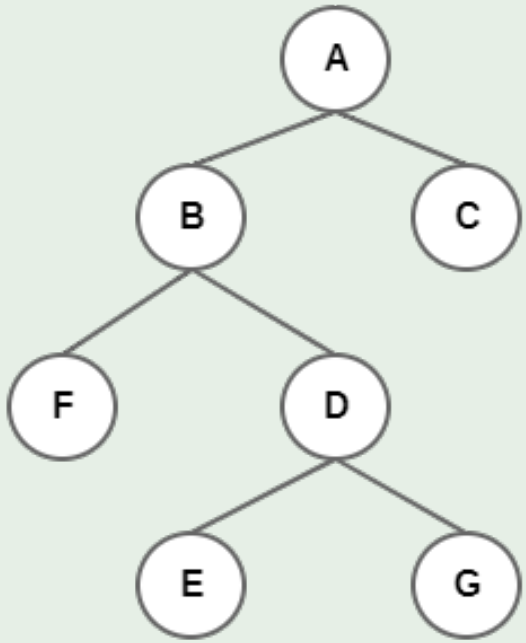


Árvore Estritamente Binária

- Cada nó possui 0 ou 2 sub-árvores;
- Nenhum nó tem filho único;
- Nós internos sempre tem 2 filhos;



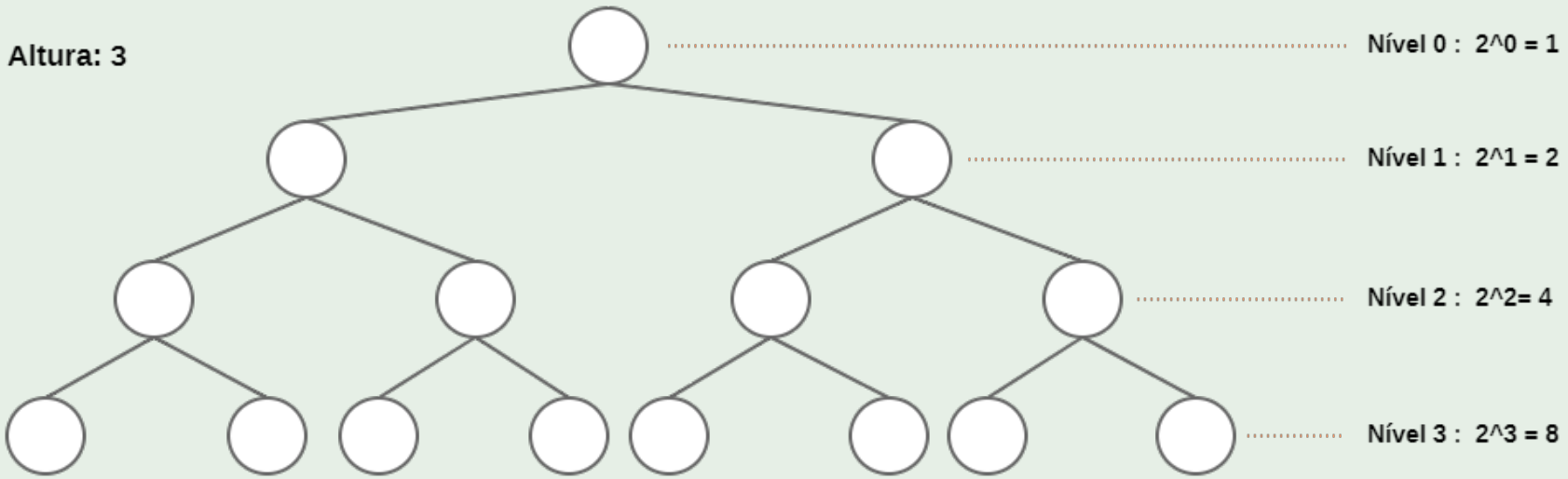
Árvore



Árvore Estritamente

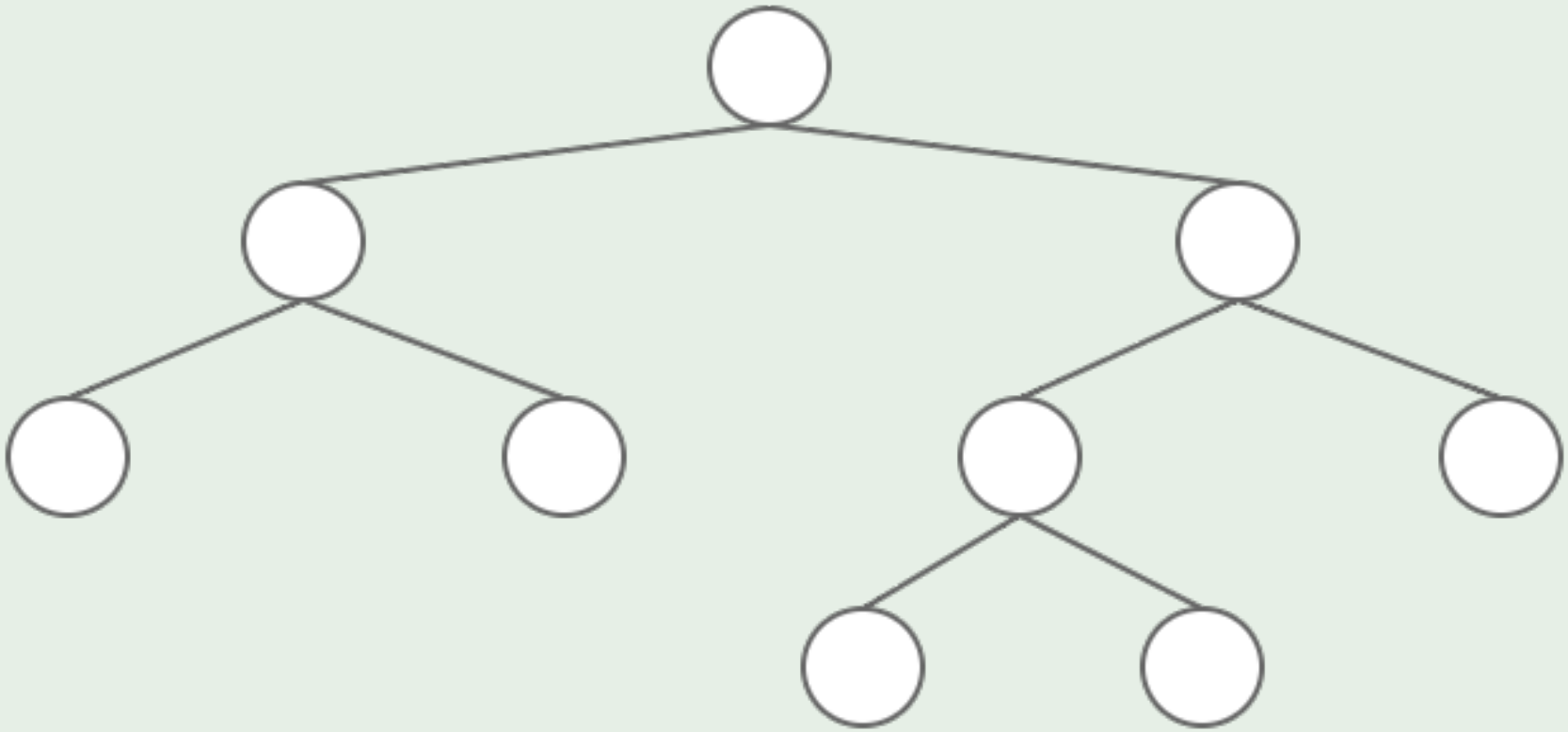
Árvore Binária Completa

- É Estritamente Binária e todos os seus nós-folha estão no mesmo nível
- O número de nós de uma árvore binária completa é $2^{h+1} - 1$, onde **h** é altura da árvore.



Árvore Binária Quase Completa

- É Estritamente Binária;
- A diferença de altura entre as sub-árvores de qualquer nó é no máximo 1.
- Se a altura da árvore é D, cada nó folha está no nível D ou D-1.

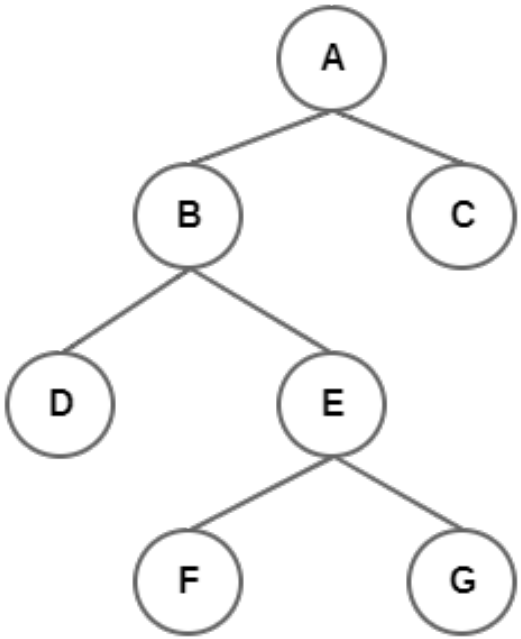


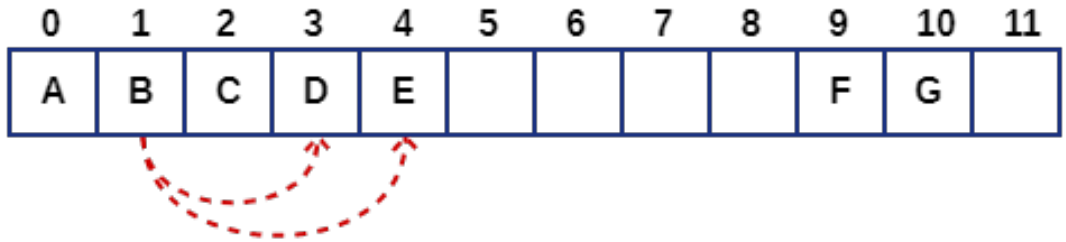
Implementando uma Árvore Binária

- Em uma árvore binária podemos realizar as seguintes operações:
  - criação de árvore;
  - inserção de um elemento;
  - remoção de um elemento;
  - acesso a um elemento;
  - destruição da árvore;
- Essas operações dependem do tipo de alocação de memória usada:
  - estática (vetor);
  - dinâmica (lista encadeada);

Alocação Estática

- uso de vetor;
- usa 2 funções para retornar a posição dos filhos à esquerda e à direita de um pai;
- $FILHO\_ESQ(PAI) = 2 * PAI + 1$;
- $FILHO\_DIR(PAI) = 2 * PAI + 2$;

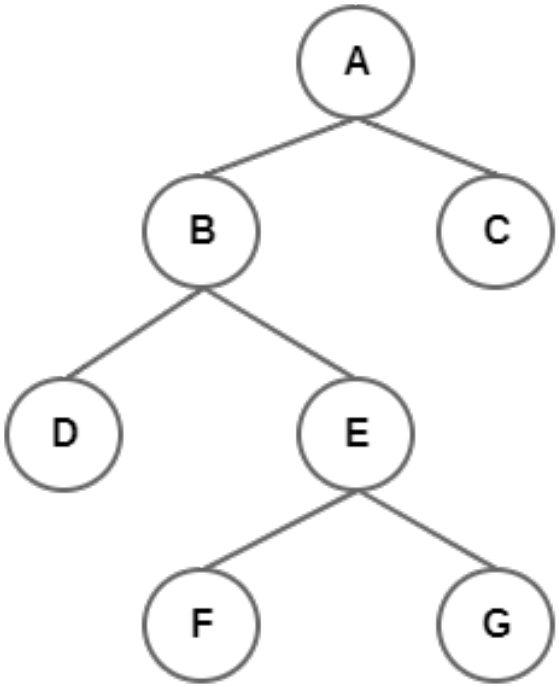


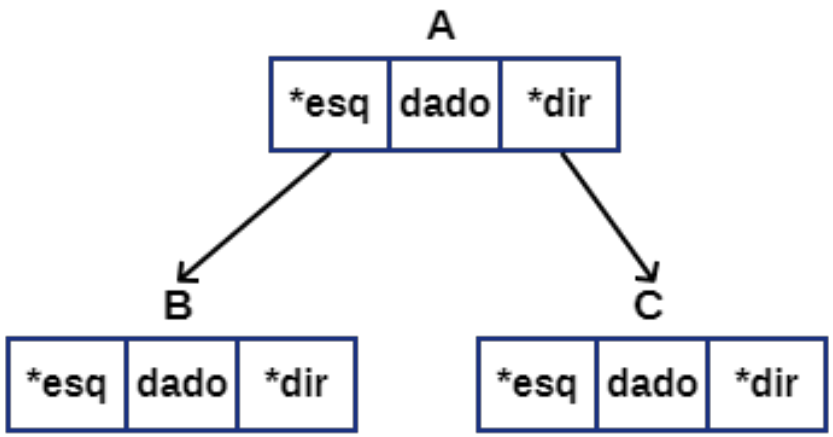


- **Quando utilizar**: Árvore Binária Completa e tamanho da árvore bem definido;

Alocação Dinâmica

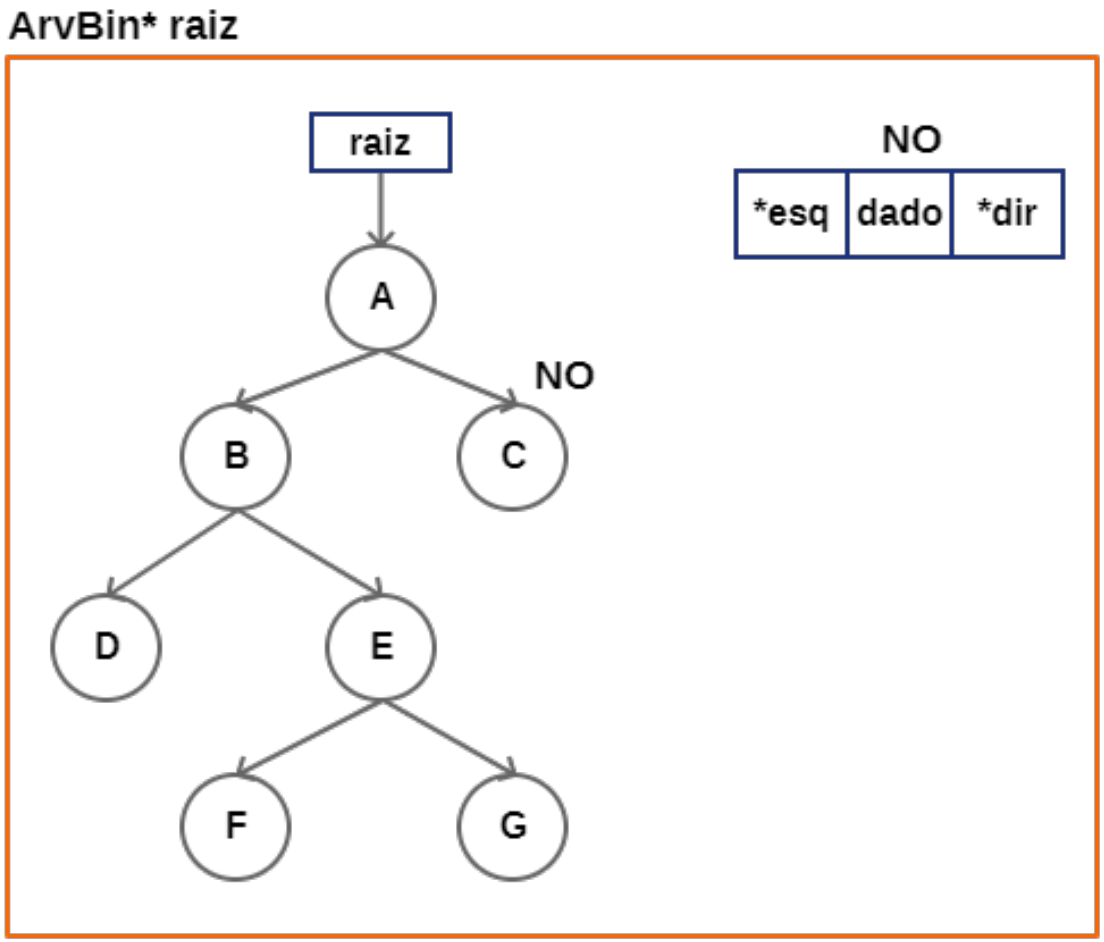
- uso de lista encadeada;
- cada nó da árvore é tratado como um ponteiro alocado dinamicamente a medida que os dados são inseridos;





Implementando uma Árvore Binária - Alocação Dinâmica

- Para guardar o primeiro nó da árvore utilizamos um ponteiro para ponteiro;
- Um ponteiro para ponteiro pode guardar o endereço de um ponteiro;
- Assim, fica fácil mudar quem é a raiz da árvore (se necessário).



Implementação em C - Alocação Dinâmica

ArvoreBinaria.h

- os protótipos das funções;
- o tipo de dado armazenado na árvore;
- o ponteiro "árvore".

ArvoreBinaria.c

- o tipo de dado "árvore";
- implementar as suas funções.

Implementação em C - Protótipos (TDA)

ArvoreBinaria.h

```
ArvBin* cria_ArvBin();
void libera_ArvBin(ArvBin *raiz);
int insere_ArvBin(ArvBin* raiz, int valor);
int remove_ArvBin(ArvBin *raiz, int valor);
int estaVazia_ArvBin(ArvBin *raiz);
int altura_ArvBin(ArvBin *raiz);
int totalNO_ArvBin(ArvBin *raiz);
int consulta_ArvBin(ArvBin *raiz, int valor);
void preOrdem_ArvBin(ArvBin *raiz);
void emOrdem_ArvBin(ArvBin *raiz);
void posOrdem_ArvBin(ArvBin *raiz);
```

Implementação em C - Definições

ArvoreBinaria.h

```c
typedef struct NO* ArvBin;
```

ArvoreBinaria.c

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

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

Função principal: main()

```c
ArvBin* raiz;
```

Implementação em C - Criar Árvore Binária

ArvoreBinaria.h

```
ArvBin* cria_ArvBin();
```

ArvoreBinaria.c

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

Função principal: main()

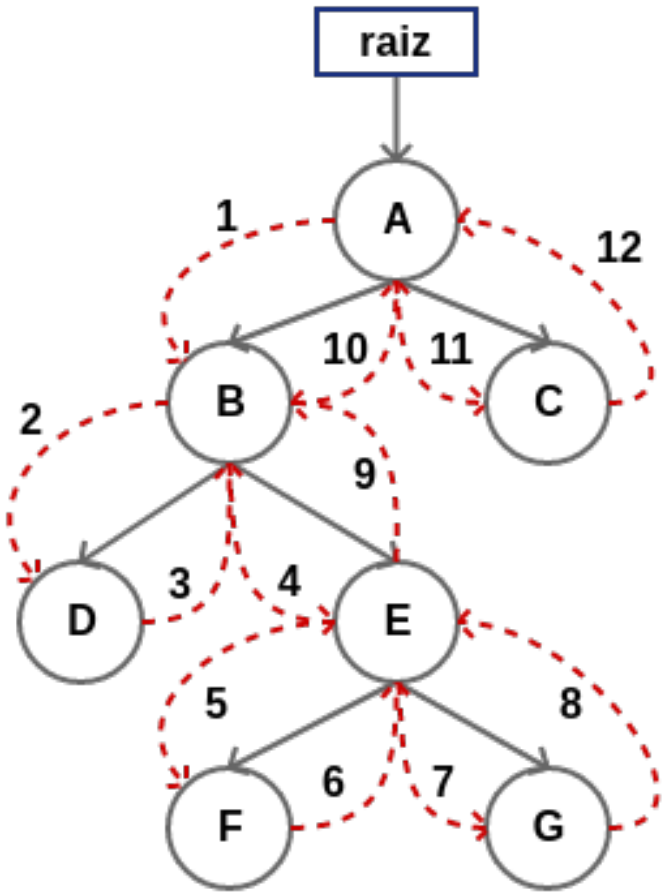
```
ArvBin* raiz = cria_ArvBin();
```

Implementação em C - Destruir Árvore Binária

ArvoreBinaria.c

```
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 }
```

Implementação em C - Destruir Árvore Binária

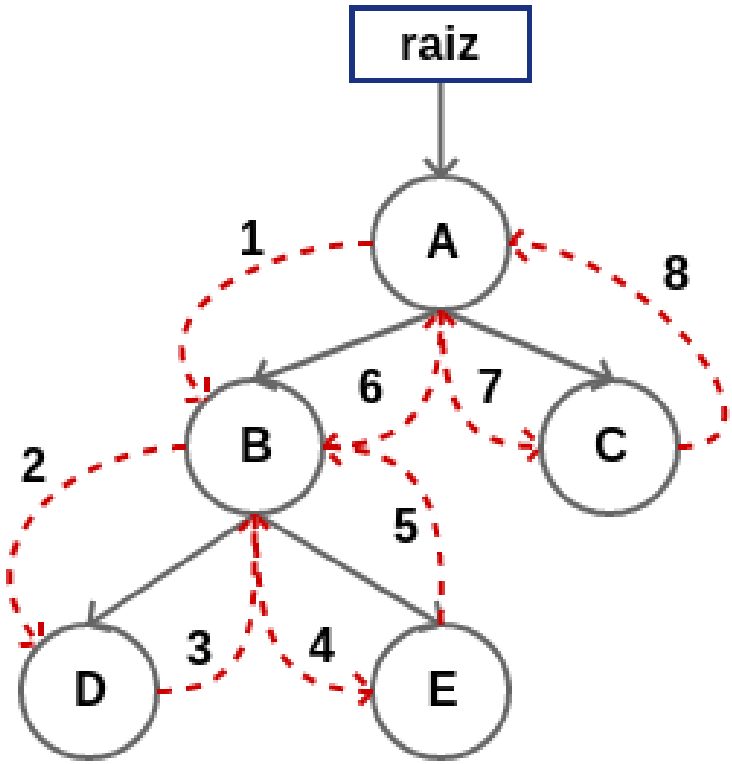


	Acesso	Ação
	1	visita B
	2	visita D
X	3	libera D, volta para B
	4	visita E
	5	visita F
X	6	libera F, volta para E
	7	visita G
X	8	libera G, volta para E
X	9	libera E, volta para B
X	10	libera B, volta para A
	11	visita C
X	12	libera C, volta para A
X		libera A

Implementação em C - Altura

```
138 int altura_ArvBin(ArvBin *raiz){
139     if (raiz == NULL)
140         return -1;
141     if (*raiz == NULL)
142         return -1;
143     int alt_esq = altura_ArvBin(&((*raiz)->esq));
144     int alt_dir = altura_ArvBin(&((*raiz)->dir));
145     if (alt_esq > alt_dir)
146         return (alt_esq + 1);
147     else
148         return(alt_dir + 1);
149 }
```

Altura - Demonstração

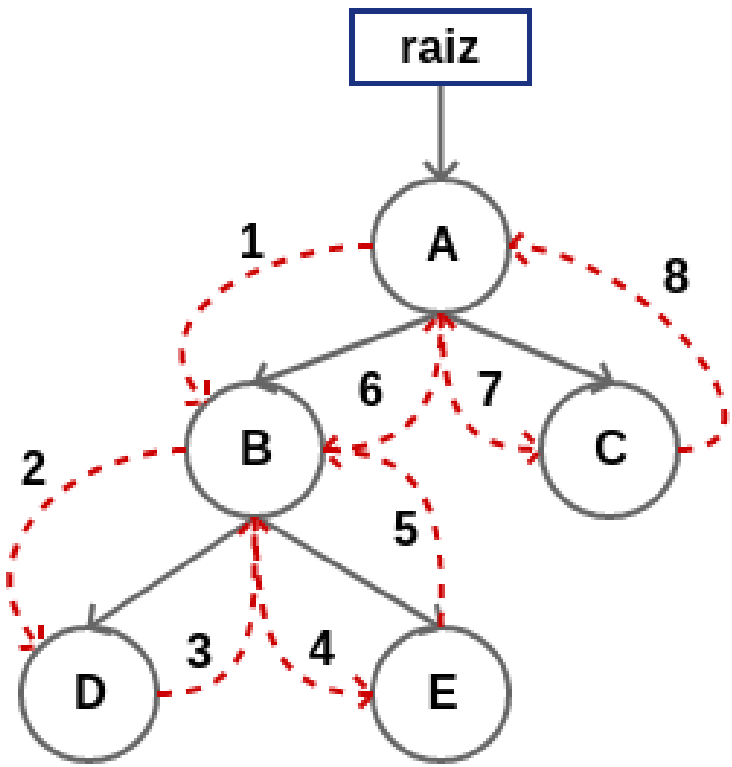


	Ação
1	visita B
2	visita D
3	D é nó folha: altura é 0. Volta para B
4	visita E
5	E é nó folha: altura é 0. Volta para B
6	Altura B é 1: maior altura dos filhos + 1. Volta para A
7	visita C
8	C é nó folha: altura é 0. Volta para A
	Altura de A é 2: maior altura dos filhos + 1

Implementação em C - Total de Nós

```c
int totalNO_ArvBin(ArvBin *raiz){
    if (raiz == NULL)
        return 0;
    if (*raiz == NULL)
        return 0;
    int alt_esq = totalNO_ArvBin(&((*raiz)->esq));
    int alt_dir = totalNO_ArvBin(&((*raiz)->dir));
    return(alt_esq + alt_dir + 1);
}
```

Total de Nós - Demonstração



	Ação
1	visita B
2	visita D
3	D é nó folha: conta como 1 nó. Volta para B
4	visita E
5	E é nó folha: conta como 1 nó. Volta para B
6	Número de nós em B é 3: total de nós a esquerda (1) + total de nós a direita (1) + 1. Volta para A
7	visita C
8	C é nó folha: conta como 1 nó. Volta para A
	Número de nós em A é 5: total de nós a esquerda (3) + total de nós a direita (1) + 1.

Percorrendo uma Árvore Binária

- Muitas operações em árvores binárias necessitam que se percorra todas os nós de suas sub-árvores, executando alguma ação ou tratamento em cada nó;
- Cada nó é visistado uma única vez.
- Isso gera uma sequência linear de nós, cuja ordem depende de como a árvore foi percorrida.

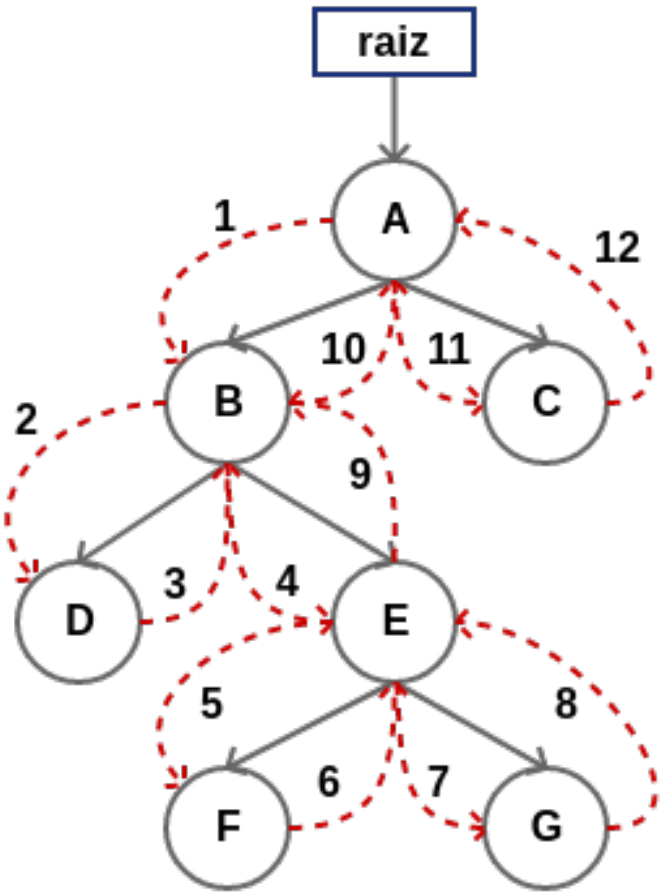
Tipos de Percurso

- **Pré-ordem**:
  - visita a **raiz**;
  - o filho da **esquerda**;
  - o filho da **direita**;
- **Em-ordem**:
  - visita o filho da **esquerda**;
  - a **raiz**;
  - o filho da **direita**
- **Pós-ordem**:
  - visita o filho da **esquerda**;
  - o filho da **direita**;
  - a **raiz**;

Percurso : Pré-Ordem

```
167 void preOrdem_ArvBin(ArvBin *raiz){
168     if(raiz == NULL)
169         return;
170     if(*raiz != NULL){
171         printf("%d\n",(*raiz)->dado);
172         preOrdem_ArvBin(&((*raiz)->esq));
173         preOrdem_ArvBin(&((*raiz)->dir));
174     }
175 }
```

Percurso : Pré-Ordem



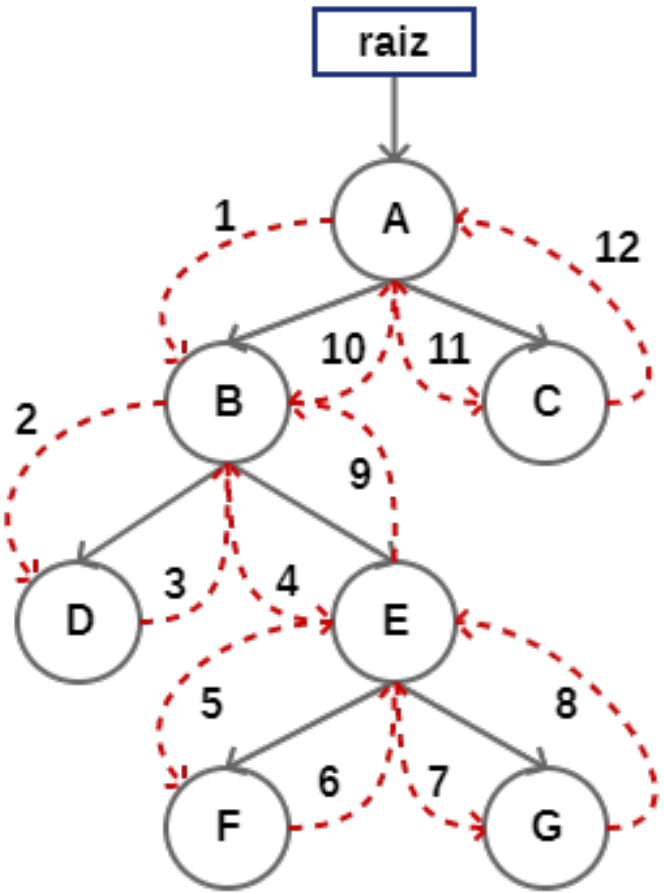
ABDEFGC

	Ação
1	imprime A, visita B
2	imprime B, visita D
3	imprime D, volta para B
4	visita E
5	imprime E, visita F
6	imprime F, volta para E
7	visita G
8	imprime G, volta para E
9	volta para B
10	volta para A
11	visita C
12	imprime C, volta para A
	visita A

Percurso : Em-Ordem

```
177 void emOrdem_ArvBin(ArvBin *raiz){
178     if(raiz == NULL)
179         return;
180     if(*raiz != NULL){
181         emOrdem_ArvBin(&((*raiz)->esq));
182         printf("%d\n",(*raiz)->dado);
183         emOrdem_ArvBin(&((*raiz)->dir));
184     }
185 }
```

Percurso : Em-Ordem



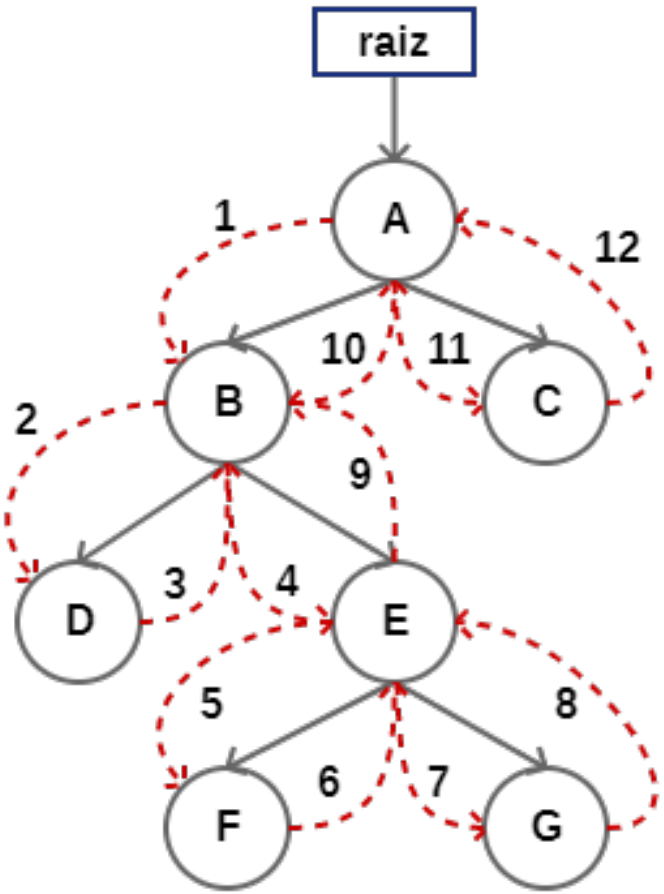
DBFEGAC

	Ação
1	visita B
2	visita D
3	imprime D, volta para B
4	imprime B, visita E
5	visita F
6	imprime F, volta para E
7	imprime E, visita G
8	imprime G, volta para E
9	volta para B
10	volta para A
11	imprime A, visita C
12	imprime C, volta para A
	visita A

Percurso : Pós-Ordem

```
187 void posOrdem_ArvBin(ArvBin *raiz){
188     if(raiz == NULL)
189         return;
190     if(*raiz != NULL){
191         posOrdem_ArvBin(&((*raiz)->esq));
192         posOrdem_ArvBin(&((*raiz)->dir));
193         printf("%d\n",(*raiz)->dado);
194     }
195 }
```

Percurso : Pós-Ordem



DFGEBCA

	Ação
1	visita B
2	visita D
3	imprime D, volta para B
4	visita E
5	visita F
6	imprime F, volta para E
7	visita G
8	imprime G, volta para E
9	imprime E, volta para B
10	imprime B, volta para A
11	visita C
12	imprime C, volta para A
	imprime A

Questões

- Quais os procedimentos para:
  - Verificar se uma árvore binária está vazia?
  - Inserir um Nó em uma árvore binária?
  - Remover um Nó em uma árvore binária?
- Pesquise e responda o que é uma **Árvore Binária de Busca**.

Referências

Referências

- André Ricardo Backes, CAPÍTULO 11 - Árvores, Editor(s): André Ricardo Backes, **Estrutura de Dados Descomplicada em Linguagem C**, Elsevier Editora Ltda., 2016, Pages 193-220, ISBN 9788535285239.