

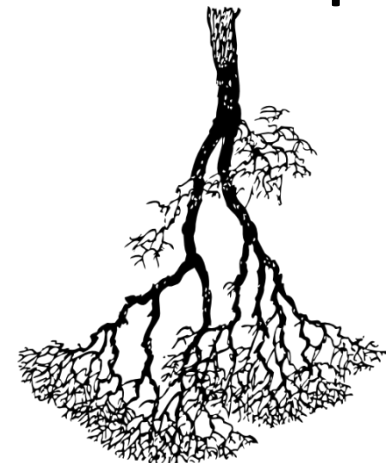
Árvore



**Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
CEFET-RJ**

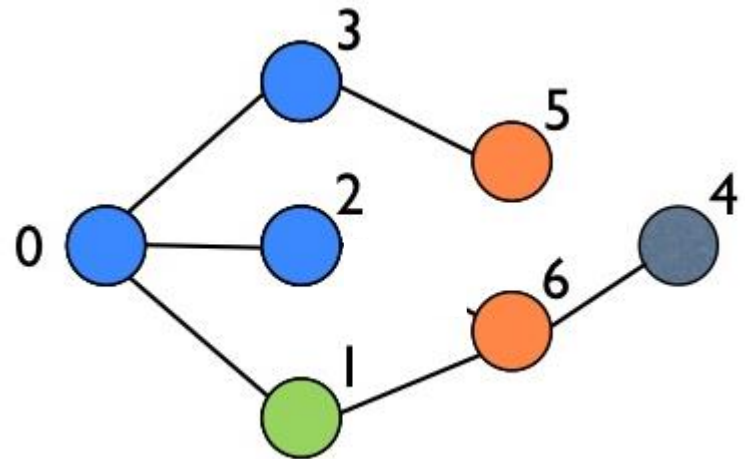
Árvore

- As listas encadeadas apresentam grande flexibilidade sobre as representações contíguas de estrutura de dados, porém sua forte característica sequencial representa o seu ponto fraco.
 - A movimentação ao longo das listas é feita alcançando um nó a cada vez.
- Muitas vezes é necessário empregar estruturas mais complexas do que as puramente sequenciais.
 - Dentre essas estruturas destacam-se as árvores.



Principais Aplicações

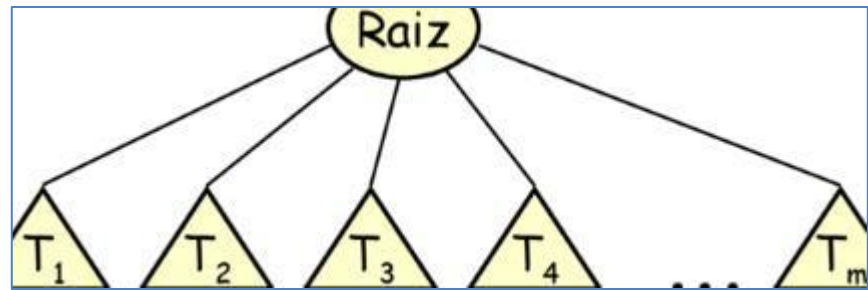
- Algoritmos de Compactação
- Algoritmos de Busca
- Aplicações em Compiladores
- Aplicações de Inteligência Artificial
- Aplicações em Computação Gráfica
- ...



Definição

- Uma árvore é um conjunto finito de n elementos denominados nós.

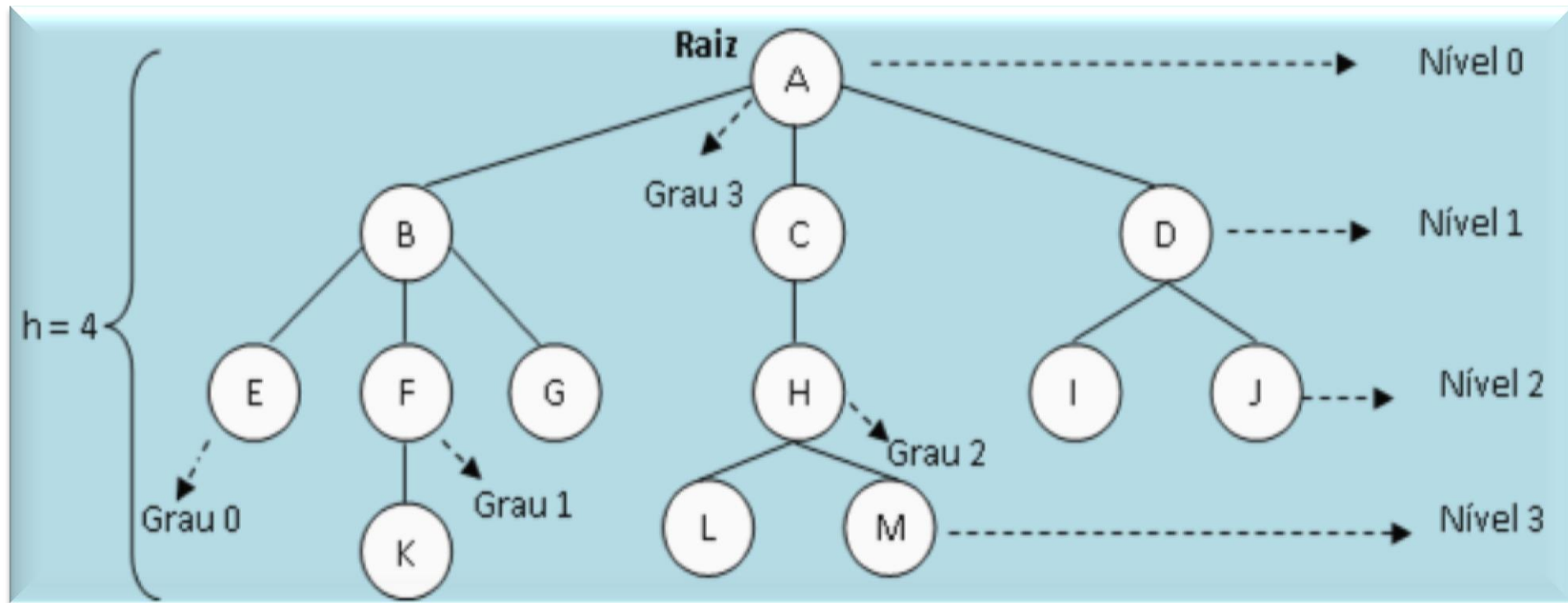
- Quando $n = 0 \Rightarrow$ árvore nula.
- Supondo $n > 0$:



- Existe um nó especial chamado **raiz**;
- Os demais nós são particionados em estruturas disjuntas de árvores (T_1, T_2, \dots, T_m) denominadas **subárvores**.
 - Como as estruturas são disjuntas, garante-se que nenhum nó não aparecerá em mais de uma subárvore.

Representação Gráfica

- A maneira mais comum de representar graficamente uma árvore é através de sua representação hierárquica.



Propriedades

- **Grau**

- representa o número de subárvores de um nó. O nó 'A' tem grau 3. Já o nó 'H' tem grau 2.

- **Folha (ou terminal)**

- nó que possui grau zero, ou seja, não possui subárvores. Os nós 'E', 'G', 'K', 'L', 'M', 'I' e 'J' são folhas.

- **Filho**

- são as raízes das subárvores de um nó, e este nó raiz é o pai delas. Os nós 'E', 'F' e 'G' são filhos do nó 'B'. Logo 'B' é o pai desses nós. E 'B' é filho do nó 'A'.

Propriedades

- **Irmãos**

- nós que possuem o mesmo pai. Os nós 'E', 'F' e 'G' são nós irmãos.

- **Nível**

- é a distância de um nó até a raiz da árvore. Como a raiz possui uma distância zero de si própria, diz-se que ela tem nível 0. No desenho, os nós 'B', 'C' e 'D' estão no nível 1 e assim sucessivamente.

- **Altura (profundidade)**

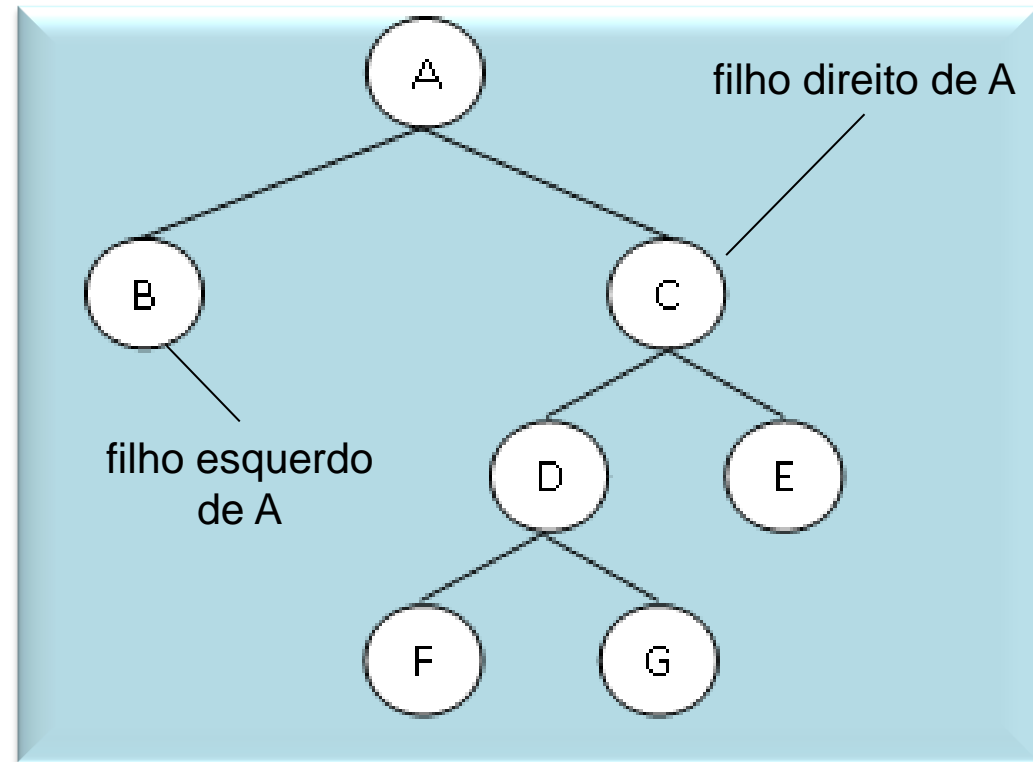
- é o nível do nó folha que tem o caminho mais longo até a raiz, somando um.
- A altura da árvore do desenho é 4.

Tipos de Árvore

- As **árvores binárias** são as estruturas com maior aplicação em computação.

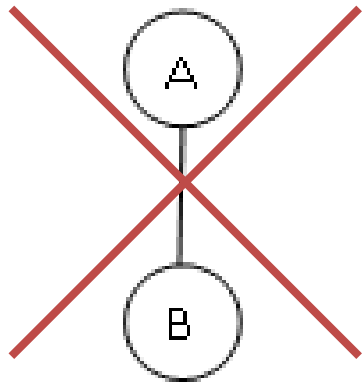
Uma árvore é binária quando os nós **NÃO** tem grau superior a 2.

- Nenhum nó tem mais do que dois filhos (ou duas subárvores).

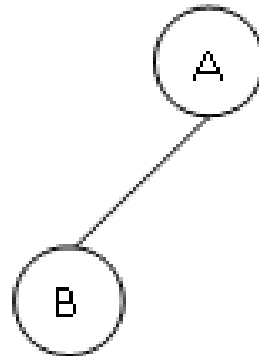


Árvore Binária

- Caso a árvore tenha apenas um filho, então deve-se indicar, graficamente, se ele é o filho direito ou esquerdo.

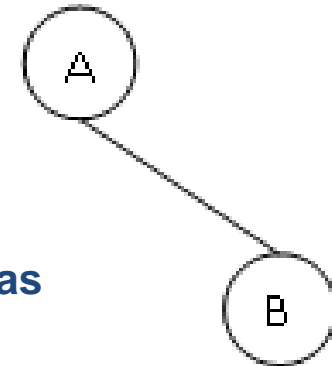


desenho incorreto
para uma árvore
binária



Filho à esquerda

Árvores Binárias

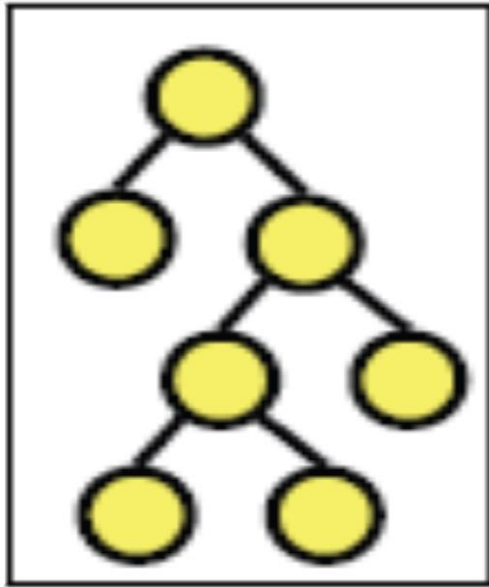


Filho à direita

Tipos de Árvores Binárias

- **Estritamente Binária**

- todo nó que não é folha possui subárvores esquerda e direita não vazias, ou seja, todo nó possui 0 ou 2 filhos.

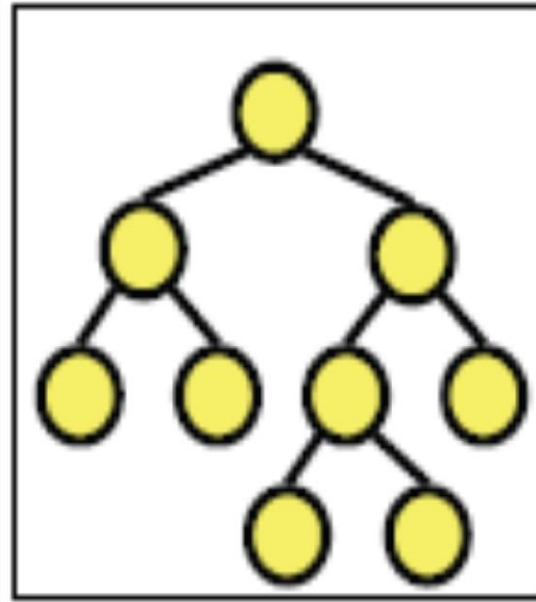


Uma árvore estritamente binária com n folhas contém sempre $2n - 1$ nós.

Tipos de Árvores Binárias

- **Binária Completa**

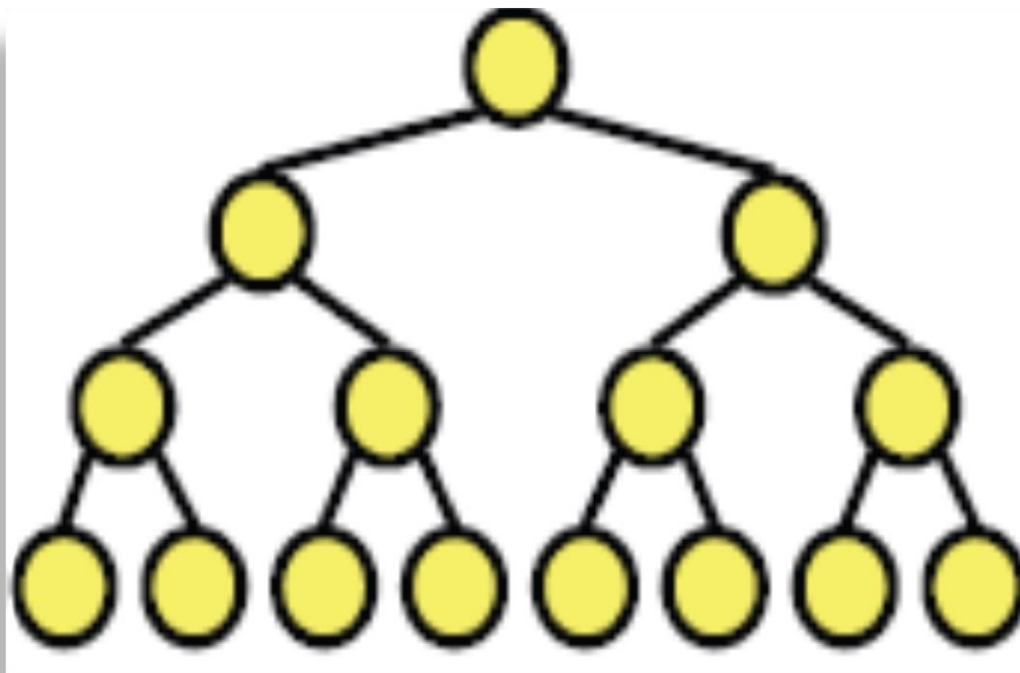
- é uma árvore estritamente binária onde todos os nós folhas encontram-se ou no último ou no penúltimo nível da árvore.



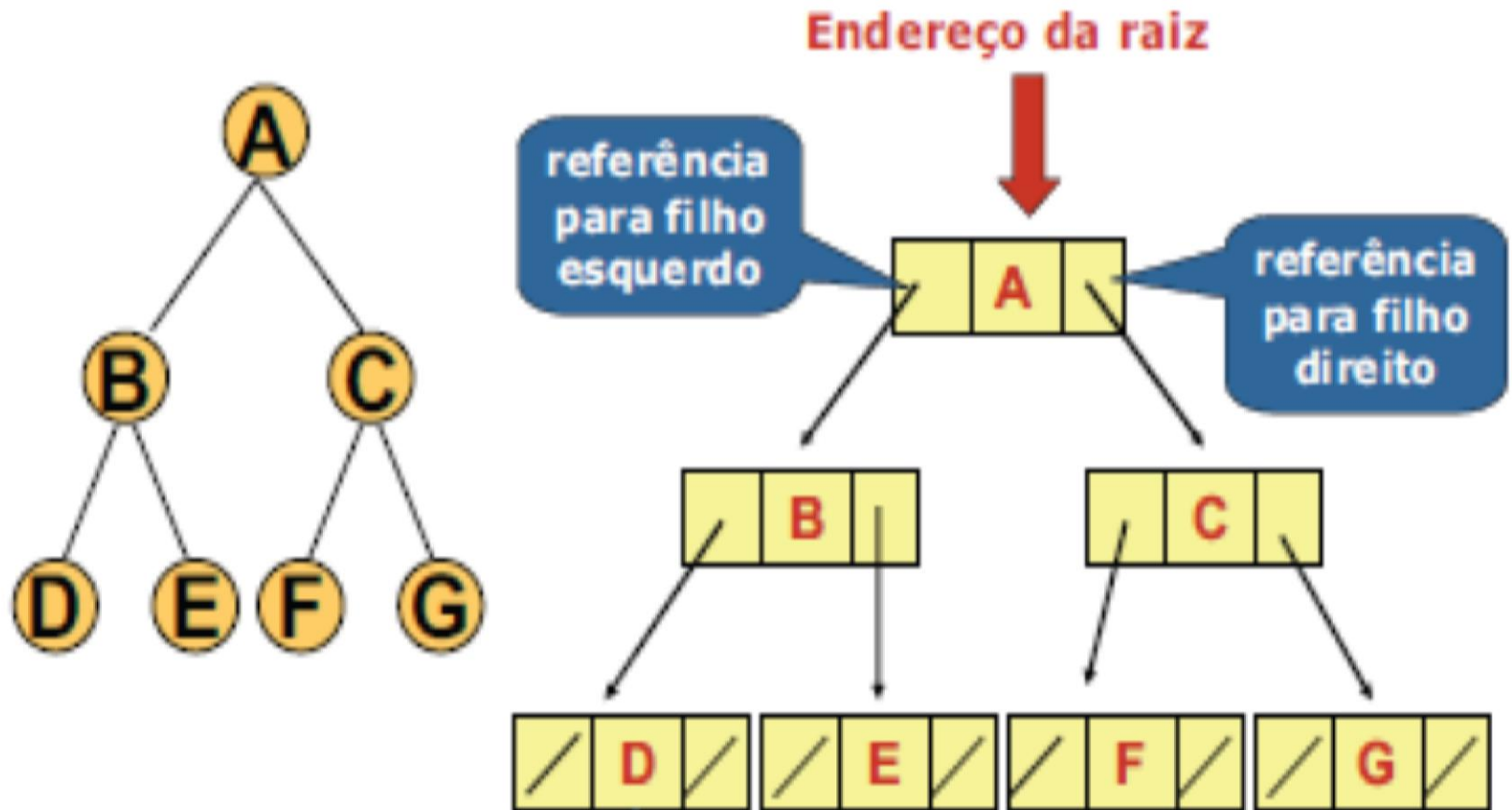
Tipos de Árvores Binárias

- **Binária Cheia**

- é uma árvore estritamente binária onde os nós folhas se encontram no último nível.



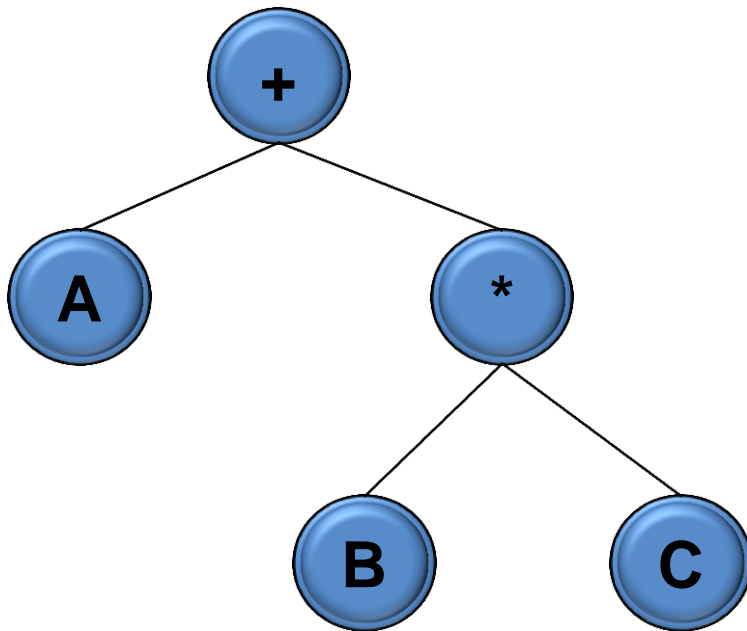
Representação



Exemplo de Árvore Binária

- Representação de expressões aritméticas

- Dada uma expressão aritmética, sua representação em árvore binária é feita de tal forma que a ordem de prioridade das operações fica implícita.



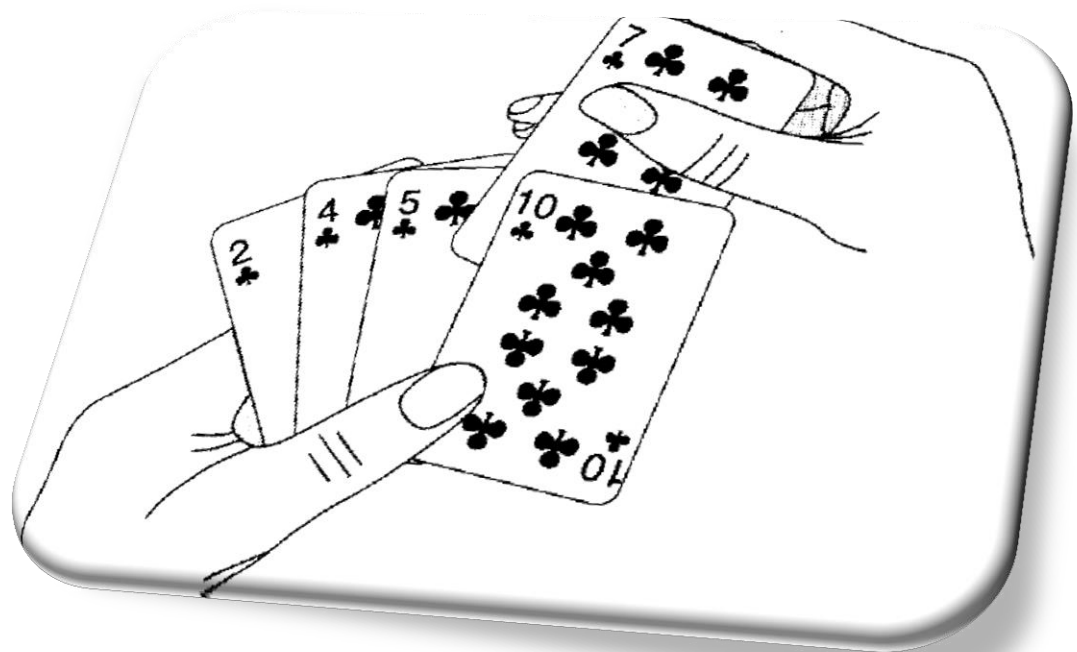
Exemplo: $A + B * C$

Regras:

- Operador de menor prioridade fica na raiz.
- A subexpressão à esquerda do operador dá origem à subárvore esquerda.
- A subexpressão à direita do operador dá origem à subárvore direita.
- Operandos sempre aparecem como folhas. Operadores nunca.

Árvore Binária de Busca

- Busca \Rightarrow operação importante em computação.
 - Array e lista encadeada \Rightarrow alto custo.
 - Para otimizar o processo: ordenação dos elementos.

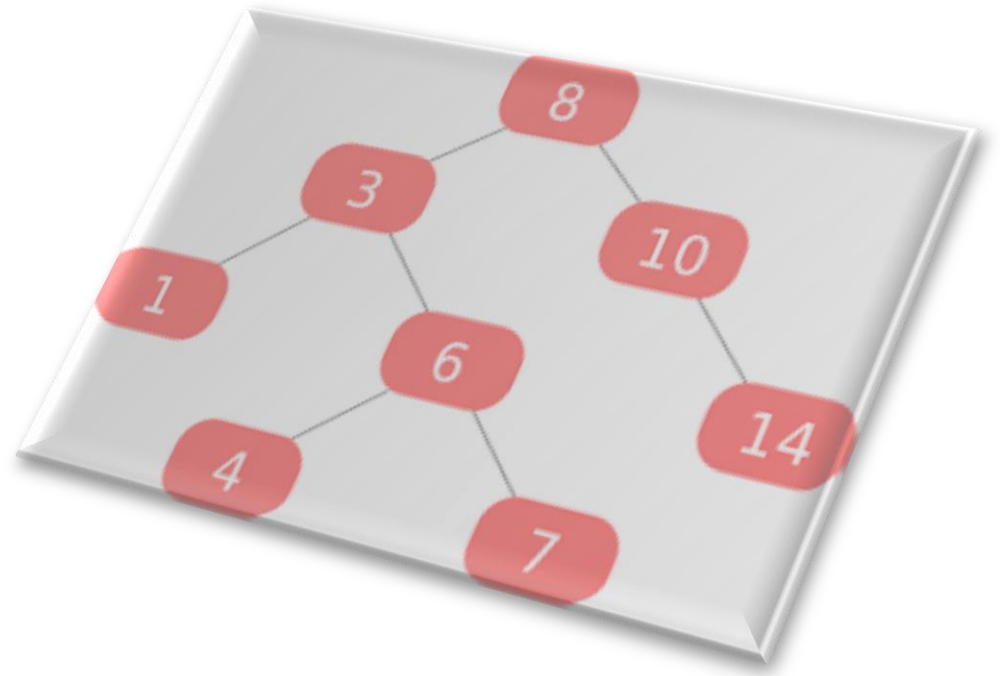


Árvore Binária de Busca

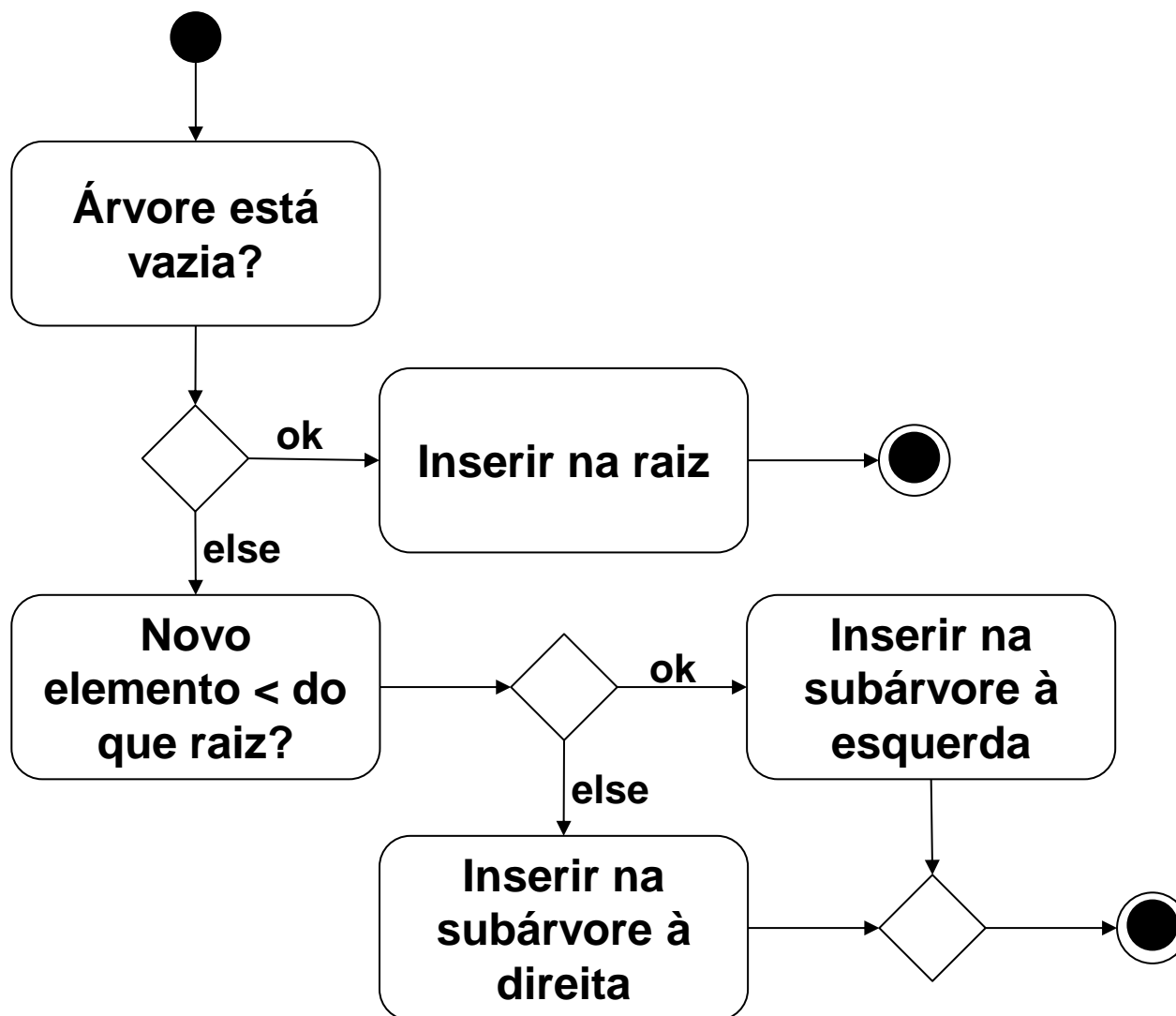
- **Um algoritmo de busca eficiente é denominado pesquisa binária.**
 - Usar array não é muito eficiente na inserção e na remoção devido à necessidade de reorganização dos elementos para a manutenção da ordenação.
 - Usar listas encadeadas resolvem o caso da inserção e de remoção (uso de ponteiros). O problema é que os algoritmos de pesquisa são sequenciais.
 - Usar árvores como estruturas de armazenamento:
 - **Árvore Binária de Busca, possui uma boa eficiência na inserção, na remoção e na pesquisa.**

Operações

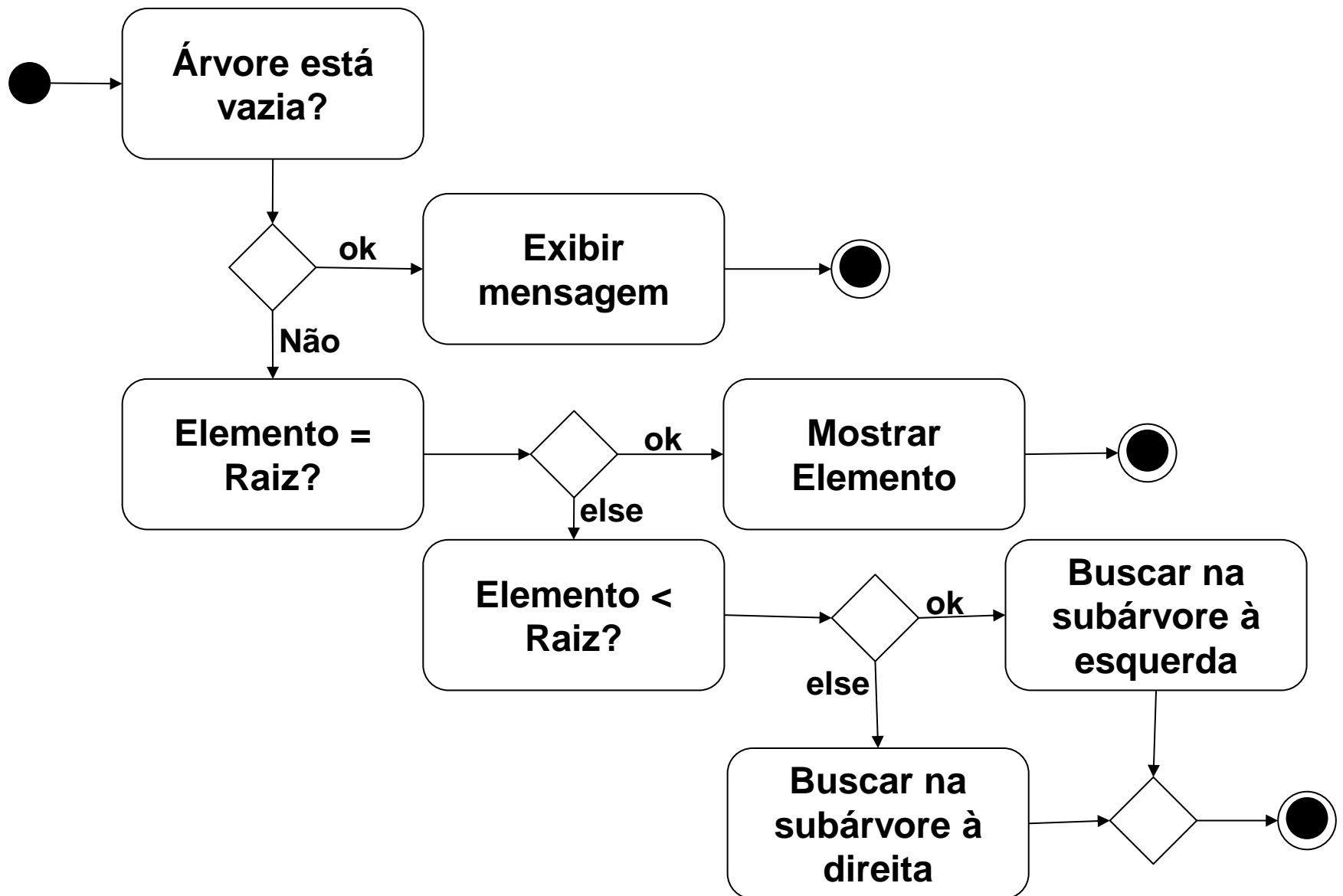
- **Árvore Binária de Busca está sempre ordenada.**
- **Principais operações:**
 - a. **inserção**
 - b. **busca**
 - c. **listagem dos nós**
 - d. **remoção**



Algoritmo para Inserir nó

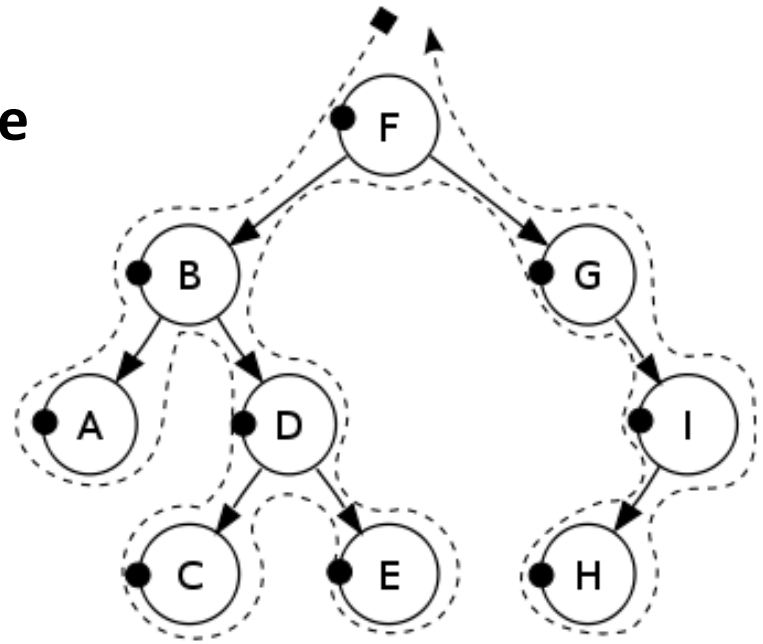


Algoritmo para Buscar nó



Listagem de elementos

- **Conceito de visita** \Rightarrow percorrer a estrutura da árvore.
- **Existem duas formas:**
 - **caminhamento em profundidade**
 - **caminhamento em largura**



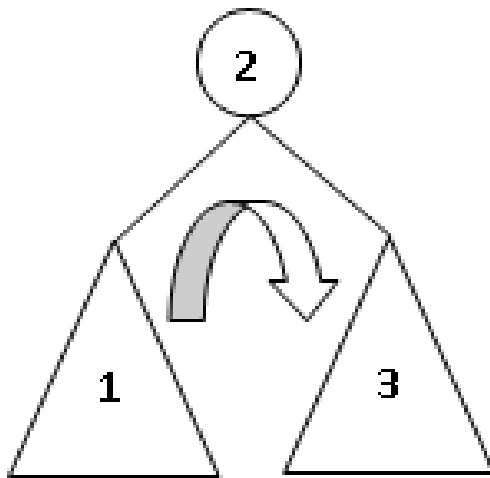
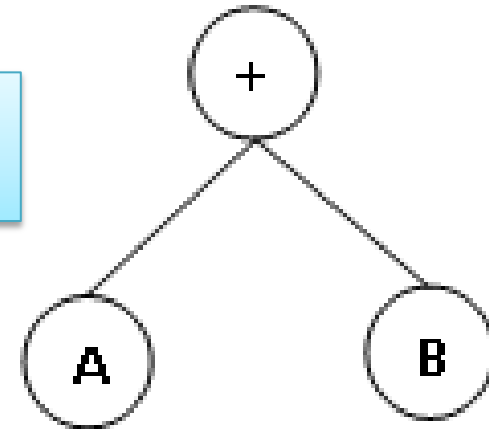
Caminhamento em Profundidade

- Todos os descendentes de um nó filho são processados antes do próximo nó filho.
- Pode ocorrer de três formas: infixa (em ordem), pré-fixa (pré-ordem) ou pós-fixa (pós-ordem).

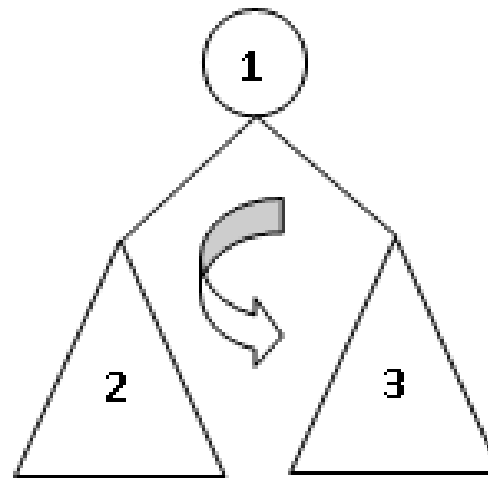
Notação	Sequência
a. Infixa (em-ordem)	Exibir a folha esquerda (E)
	Exibir a raiz (R)
	Exibir a folha direita (D)
b. Prefixa (pré-ordem)	Exibir a raiz (R)
	Exibir a folha a esquerda (E)
	Exibir a folha a direita (D)
c. Pósfixa (pós-ordem)	Exibir a folha a esquerda (E)
	Exibir a folha a direita (D)
	Exibir a raiz (R)

Caminhamento em Profundidade

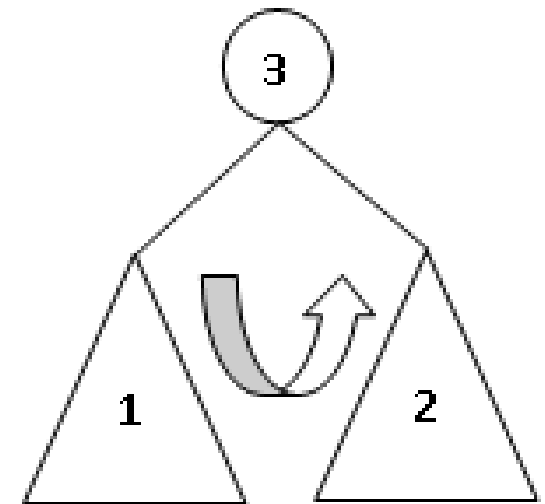
Observe a árvore binária da expressão aritmética: $A + B$.



(a)



(b)



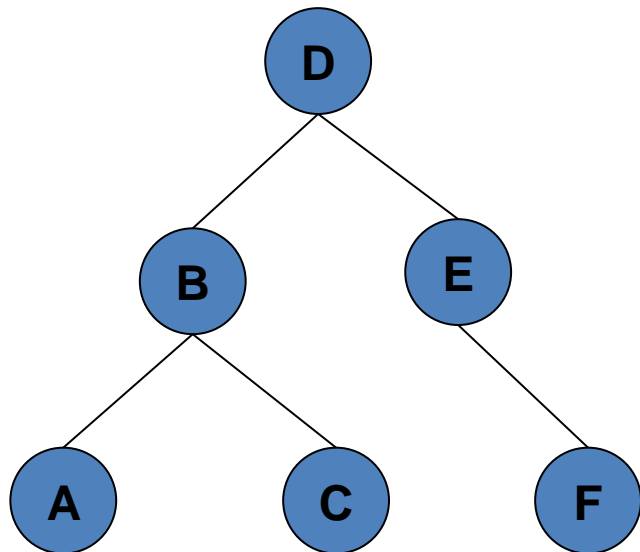
(c)

Caminhamento em Profundidade

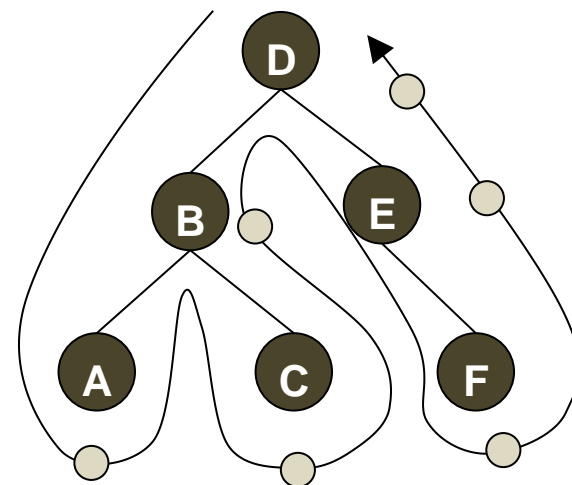
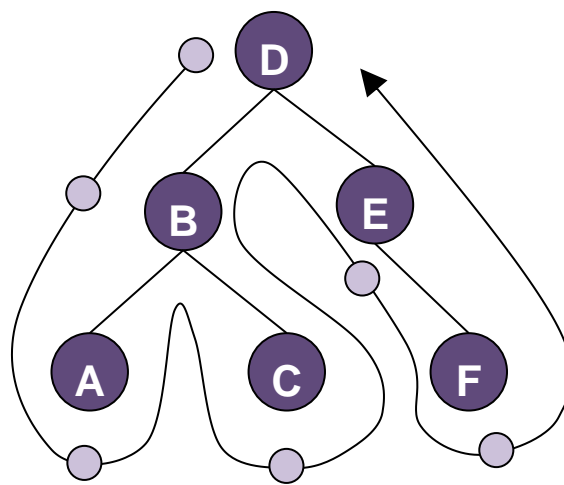
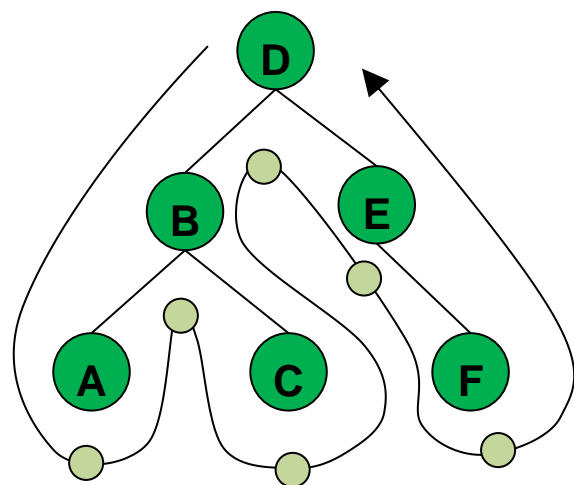
- Se existe a possibilidade de processar os nós de uma árvore binária de busca em ordem crescente, pode-se usar o passeio em-ordem.
- Considerando a árvore aritmética anterior, pode-se verificar:

Formas de passeio	Saídas
Em-ordem	A + B
Pré-ordem	+ A B
Pós-ordem	A B +

Exemplos



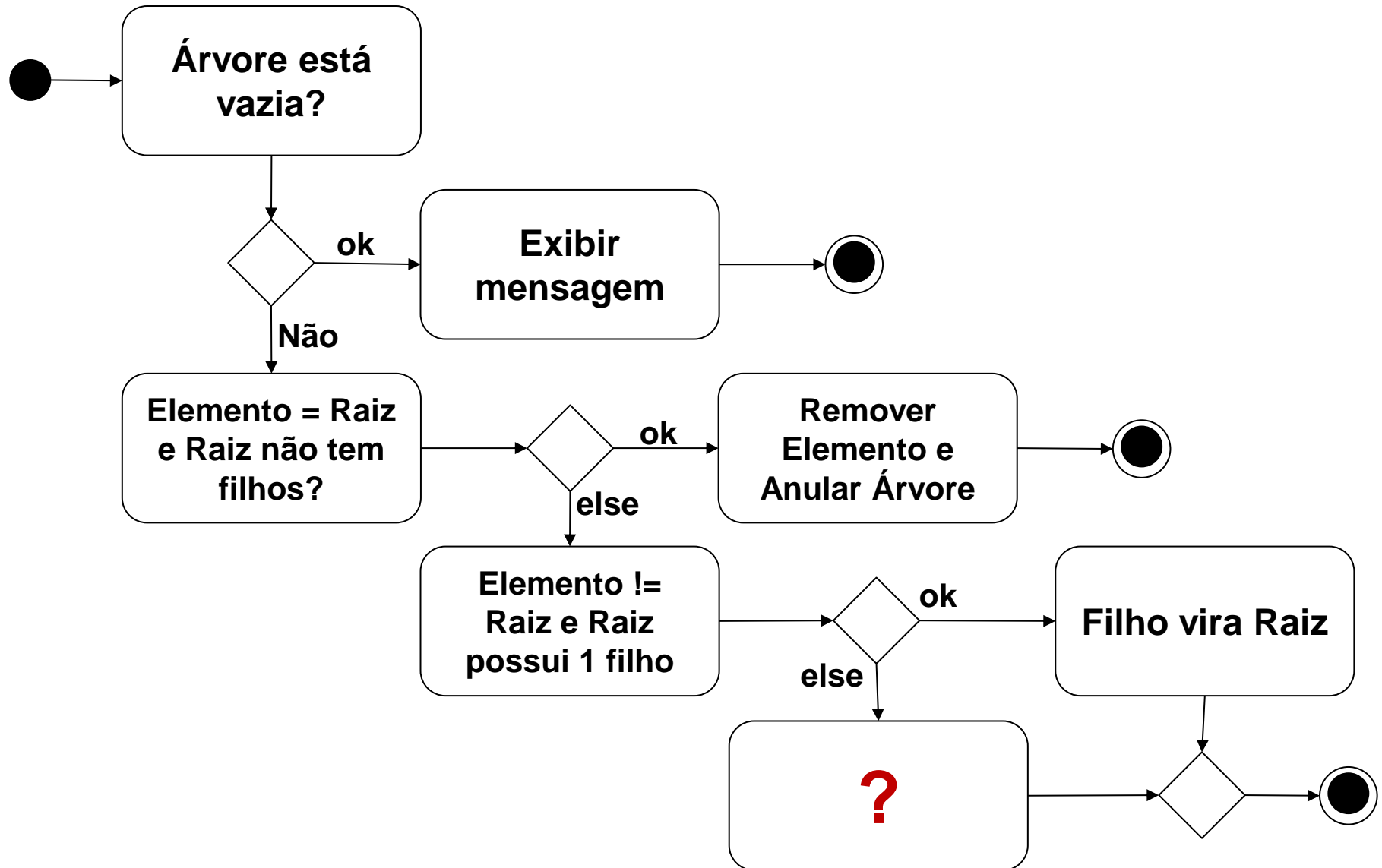
Formas de passeio	Saídas
Em-ordem	A B C D E F
Pré-ordem	D B A C E F
Pós-ordem	A C B F E D



Caminhamento em Largura

- **Ocorre de forma horizontal:**
 - da raiz para todos os nós filhos, depois para os filhos desses nós e assim sucessivamente.
- **Cada nível da árvore é pesquisado antes que o próximo nível seja iniciado.**
- **O passeio em largura ocorre na forma denominada em-nível.**

Algoritmo para Remover nó



Remoção - raiz possui mais de um filho

- Não existe a possibilidade dos filhos da subárvore esquerda serem maiores ou iguais à raiz; e na subárvore direita, não podem existir filhos menores do que a raiz.

Identificar o maior elemento da subárvore esquerda e o posicionar na raiz, ou pegar o menor elemento da subárvore a direita.

A árvore obtida continua seguindo a definição, ou seja, continua ordenada.

Implementação da Árvore Binária de Busca

```
#include <iostream>
```

```
using namespace std;
```

```
struct no {  
    int info;  
    struct no * esq, * dir;  
};
```

```
typedef struct no * noPtr;
```

```
int menu();
```

```
int menu2();
```

```
void inserir(noPtr *, int);
```

```
void remover(noPtr *, int);
```

```
bool buscar(noPtr, int);
```

```
noPtr maior(noPtr *);
```

```
void listarEmOrdem(noPtr);
```

```
void listarPreOrdem(noPtr);
```

```
void listarPosOrdem(noPtr);
```

```
bool arvoreVazia(noPtr);
```

Função Principal

```
main() {  
    int op1, op2, x;  bool achei;  noPtr raiz = NULL;  
    do {  
        op1 = menu();  
        switch(op1) {  
            case 1:  cout << "\nDigite o elemento que voce deseja inserir: ";  
                     cin >> x;  
                     inserir(&raiz, x); break;  
            case 2:  cout << "\nDigite o elemento que voce deseja remover: ";  
                     cin >> x;  
                     remover (&raiz, x); break;  
            case 3:  op2 = menu2();  
                     if (op2 == 1) listarEmOrdem(raiz);  
                     if (op2 == 2) listarPreOrdem(raiz);  
                     if (op2 == 3) listarPosOrdem(raiz); break;  
            case 4:  cout << "\nDigite o elemento que voce deseja consultar: ";  
                     cin >> x;  
                     achei = buscar(raiz, x);  
                     if (!achei)  
                         cout << "Elemento nao encontrado" << endl; break;  
        }  
    } while(op1 != 5);  
}
```

Menus

```
int menu()
{
    int opcao;
    cout << "\n\n\n---- Menu Principal ----\n\n"
        << "\n1.Inserir no na arvore"
        << "\n2.Remover no na arvore"
        << "\n3.Listar todos os nos da arvore"
        << "\n4.Buscar no"
        << "\n5.Sair"
        << "\nDigite uma opcao: ";
    cin >> opcao;
    return opcao;
}
```

```
int menu2()
{
    int opcao;
    cout << "\n\nTipos de listagem:"
        << "\n\t1.Em Ordem"
        << "\n\t2.Pre Ordem"
        << "\n\t3.Pos Ordem"
        << "\n\nEscolha o tipo de listagem: ";
    cin >> opcao;
    return opcao;
}
```

Função para Inserir Elemento e Função para Verificar Árvore Vazia

```
void inserir(noPtr * p, int x)
{
    if (arvoreVazia(*p))
    {
        *p = new no;
        (*p)->info = x;
        (*p)->esq = NULL;
        (*p)->dir = NULL;
    }
    else
    {
        if (x < ((*p)->info))
            inserir(&((*p)->esq), x);
        else
            inserir(&((*p)->dir), x);
    }
}
```

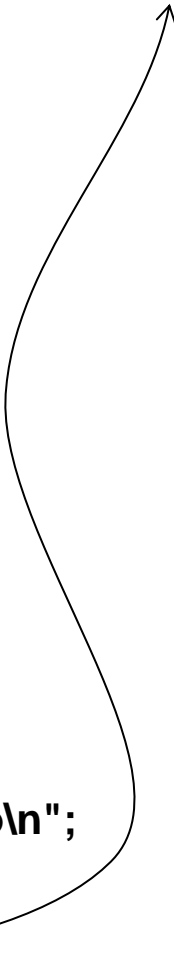
```
bool arvoreVazia(noPtr p)
{
    if (p)
        return false;
    else
        return true;
}
```

Função para Buscar Elemento

```
bool buscar(noPtr p, int x)
{
    bool achei = false;
    if(arvoreVazia(p))
        cout << "\nArvore vazia " << endl;
    else
    {
        if (x == (p->info))
        {
            cout << "\nO elemento: " << p->info << " foi encontrado na arvore: \n";
            achei = true;
        }
        else
        {
            if (x < (p->info))
                buscar((p->esq), x);
            else
                buscar((p->dir), x);
        }
    }
    return achei;
}
```


Função para Remover Elemento

```
void remover(noPtr *p, int x){
    noPtr aux;
    if (arvoreVazia(*p))
        cout << "\nArvore vazia" << endl;
    else {
        if (x == ((*p)->info)) {
            aux = *p;
            if (((*p)->esq) == NULL)
                *p = (*p)->dir;
            else
                if (((*p)->dir) == NULL)
                    *p = (*p)->esq;
                else {
                    aux = maior(&((*p)->esq));
                    (*p)->info = aux->info;
                }
            delete(aux);
            cout << "\nO elemento foi removido\n";
        }
        else
            if ((x < ((*p)->info)))
                remover(&((*p)->esq), x);
            else
                remover(&((*p)->dir), x);
    }
}
```



The diagram illustrates a recursive call. A curved arrow originates from the 'else' branch of the first 'if' statement (the one checking for equality of x and node info) and points to the 'else' branch of the second 'if' statement (the one comparing x with the node's info to decide which subtree to recurse into).

Função para verificar qual é o maior Elemento da Subárvore à esquerda (Remoção)

```
noPtr maior(noPtr *p) {  
    noPtr t;  
    t = *p;  
    if ((t->dir) == NULL)  
    {  
        *p = (*p)->esq;  
        return(t);  
    }  
    else  
        return (maior(&((*p)->dir)));  
}
```

Funções para Listar Elementos

```
void listarEmOrdem(noPtr p) {  
    if (!arvoreVazia(p)) {  
        listarEmOrdem(p->esq);  
        cout << "\t" << p->info;  
        listarEmOrdem(p->dir);  
    }  
}
```

```
void listarPosOrdem(noPtr p) {  
    if (!arvoreVazia(p)) {  
        listarPosOrdem(p->esq);  
        listarPosOrdem(p->dir);  
        cout << "\t" << p->info;  
    }  
}
```

```
void listarPreOrdem(noPtr p) {  
    if (!arvoreVazia(p)) {  
        cout << "\t" << p->info;  
        listarPreOrdem(p->esq);  
        listarPreOrdem(p->dir);  
    }  
}
```

Referências

- **Moraes. *Estruturas de Dados e Algoritmos – uma abordagem didática*. Ed. Futura**
- **Markenzon e Szwarcfiter. *Estruturas de Dados e seus Algoritmos*. Ed. LTC**
- **Deitel. *Como Programar em C/C++*. Ed. Pearson**