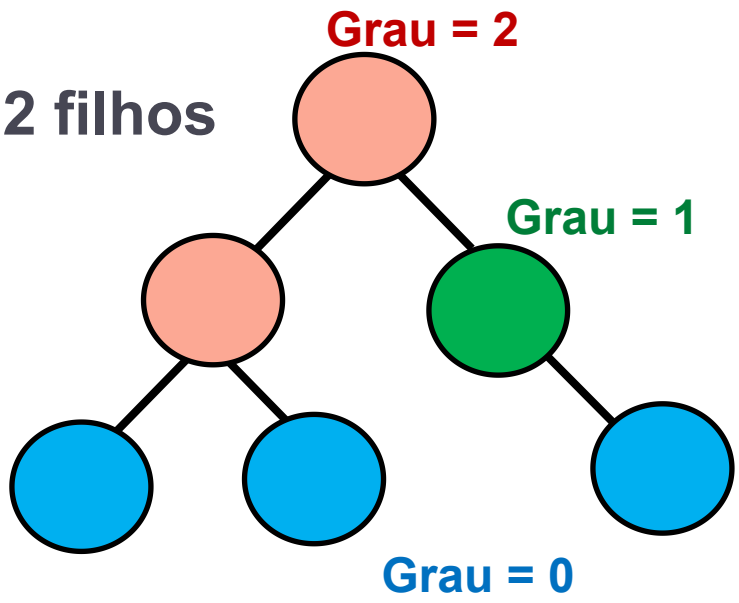
Árvores Binárias

Prof. Luiz Gustavo Almeida Martins

Árvores binárias: introdução

Árvore com **grau 2** (aridade)

Nós da árvore podem ter **0, 1 ou 2 filhos**



Árvores binárias: introdução

Árvore com **grau 2** (aridade)

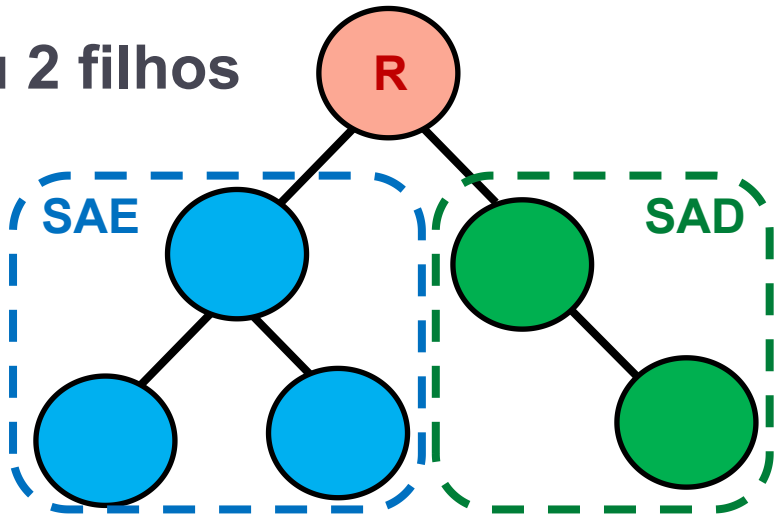
Nós da árvore podem ter **0, 1 ou 2 filhos**

Representação:

Nó raiz (R)

Subárvore da esquerda (SAE)

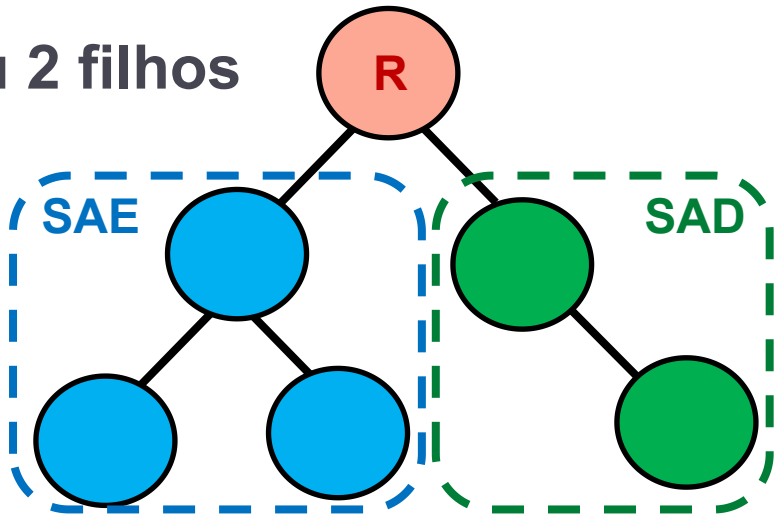
Subárvore da direita (SAD)



Árvores binárias: introdução

Árvore com **grau 2** (aridade)

Nós da árvore podem ter **0, 1 ou 2 filhos**



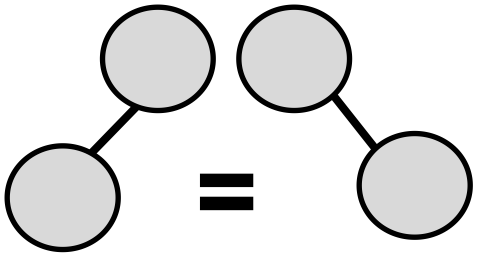
Representação:

Nó raiz (R)

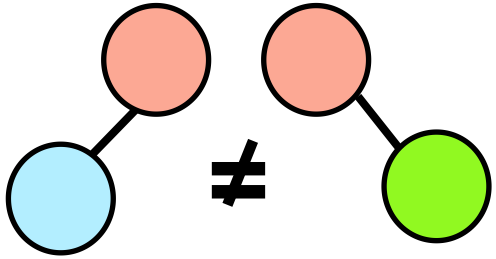
Subárvore da esquerda (SAE)

Subárvore da direita (SAD)

SAE ≠ SAD



Árvore qualquer

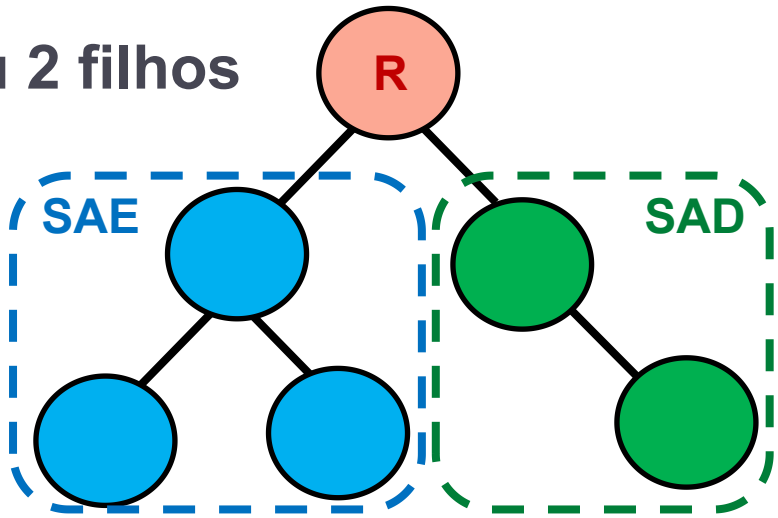


Árvore binária

Árvores binárias: introdução

Árvore com **grau 2** (aridade)

Nós da árvore podem ter **0, 1 ou 2 filhos**



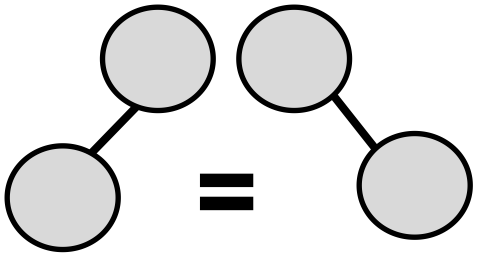
Representação:

Nó raiz (R)

Subárvore da esquerda (SAE)

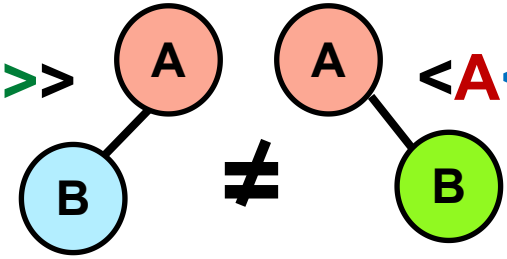
Subárvore da direita (SAD)

SAE ≠ SAD



Árvore qualquer

<A<B<><><>><>>

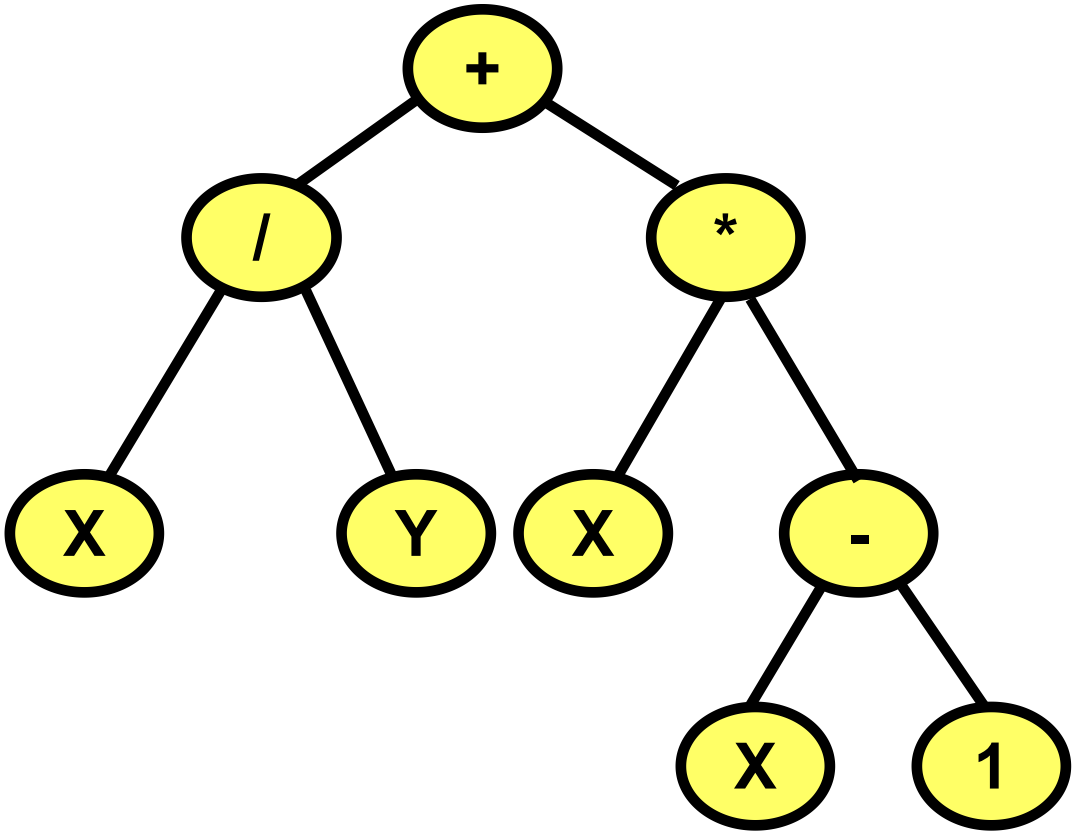


<A<><B<><><>>>

Árvore binária

Exemplo de árvore binária

Expressão aritmética: (X/Y) + (X * (X-1))



Árvores binárias: propriedades

Altura:

Altura **máxima**: $h_{max} = n-1$

Nº de nós de uma árvore binária (AB) de altura ***h***:

Número **mínimo** de nós: $n_{min} = h+1$

Árvores binárias: propriedades

Altura:

Altura **máxima**: $h_{max} = n-1$

Altura **mínima**: $h_{min} = log_2\ n\ (truncado)$

Nº de nós de uma árvore binária (AB) de altura ***h***:

Número **mínimo** de nós: $n_{min} = h+1$

Número **máximo** de nós: $n_{max} = 2^{(h+1)} - 1$

Árvores binárias: propriedades

Altura:

Altura **máxima**: $h_{max} = n-1$

Altura **mínima**: $h_{min} = log_2\ n$ *(truncado)*

Nº de nós de uma árvore binária (AB) de altura ***h***:

Número **mínimo** de nós: $n_{min} = h+1$

Número **máximo** de nós: $n_{max} = 2^{(h+1)} - 1$

Número **máximo** de nós de um **nível *X*** é 2^X

Árvore binária: percorrimento

É importante estabelecer uma **forma sistemática de visitar os elementos** de uma estrutura de dados

Árvore binária: percorrimento

É importante estabelecer uma **forma sistemática de visitar os elementos** de uma estrutura de dados

- Essa tarefa é trivial em **estruturas lineares**

Árvore binária: percorrimento

É importante estabelecer uma **forma sistemática de visitar os elementos** de uma estrutura de dados

Essa tarefa é trivial em **estruturas lineares**

Em **estruturas espaciais** existem diferentes formas que diferem em relação à **ordem** em que o percurso é realizado:

- Pré-ordem (*preorder*)
- Simétrica ou em ordem (*in order*)
- Pós-ordem (*postorder*)
- Por nível

Árvore binária: percorrimento

É importante estabelecer uma **forma sistemática de visitar os elementos** de uma estrutura de dados

Essa tarefa é trivial em **estruturas lineares**

Em **estruturas espaciais** existem diferentes formas que diferem em relação à **ordem** em que o percurso é realizado:

- Pré-ordem (*preorder*)
- Simétrica ou em ordem (*in order*)
- Pós-ordem (*postorder*)

} **busca em profundidade**

- Por nível ← **busca em largura**

Árvore binária: percorrimento

É importante estabelecer uma **forma sistemática de visitar os elementos** de uma estrutura de dados

Essa tarefa é trivial em **estruturas lineares**

Em **estruturas espaciais** existem diferentes formas que diferem em relação à **ordem** em que o percurso é realizado:

- Pré-ordem (*preorder*)
- Simétrica ou em ordem (*in order*)
- Pós-ordem (*postorder*)

} **busca em profundidade**

- Por nível ← **busca em largura**

Cada ordem de percurso pode ser adequada a um tipo diferente de aplicação

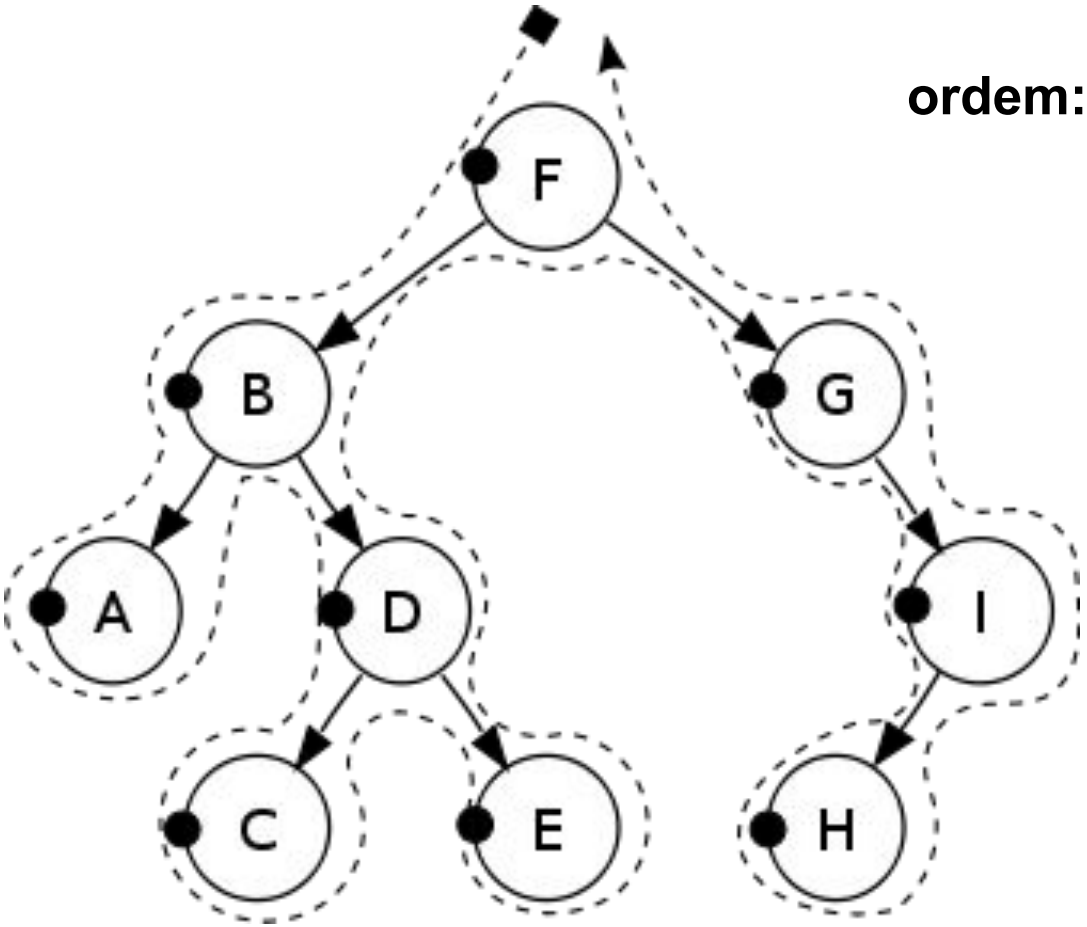
Árvore binária: percorrimento

Busca em profundidade:

Pré-ordem: trata nó raiz, percorre subárvore a esquerda, e percorre subárvore a direita

Ex: exibir o conteúdo de uma árvore (notação textual)

Exemplo: percorrimento pré-ordem



ordem: F, B, A, D, C, E, G, I, H

fonte: wikipedia

Árvore binária: percorrimento

Busca em profundidade:

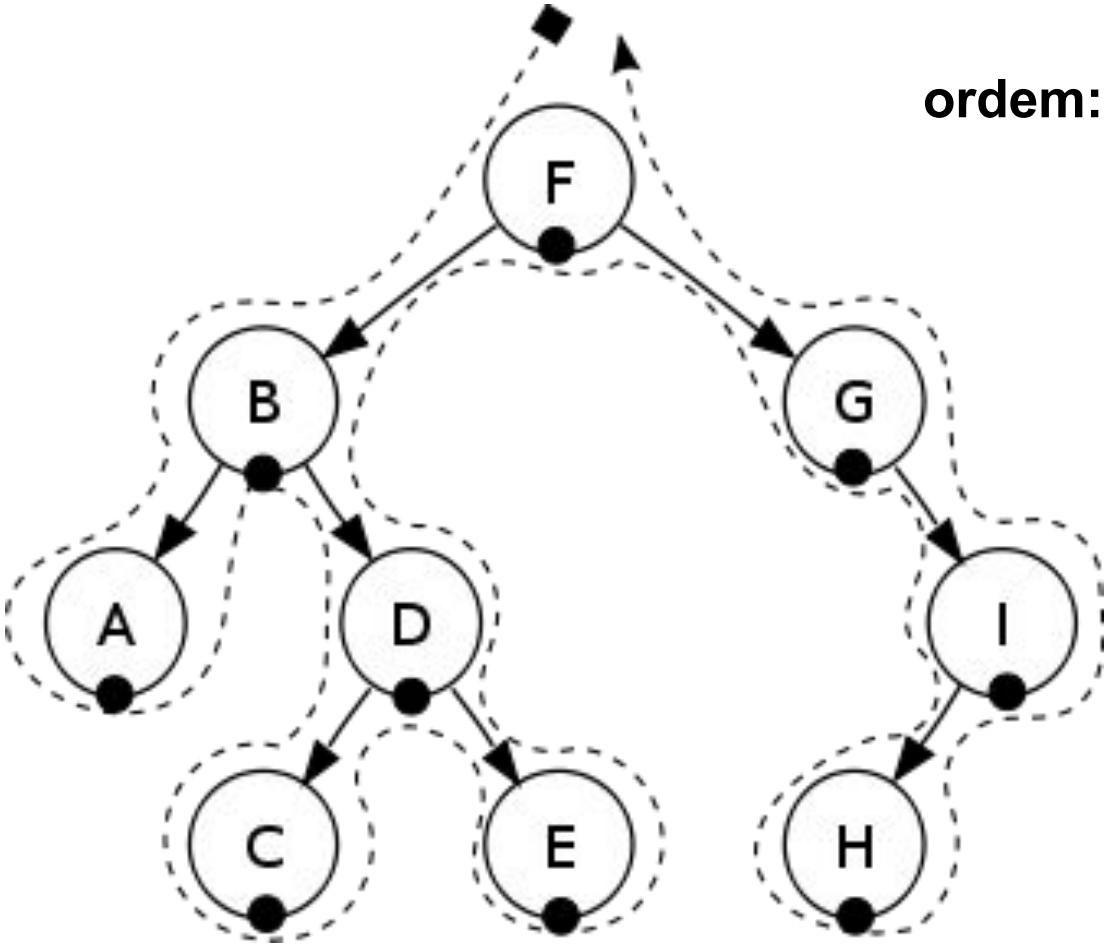
Pré-ordem: trata nó raiz, percorre subárvore a esquerda, e percorre subárvore a direita

Ex: exibir o conteúdo de uma árvore (notação textual)

Em-ordem (ordem simétrica): percorre subárvore a esquerda, trata nó raiz, e percorre subárvore a direita

Ex: exibir uma expressão matemática representada em árvore

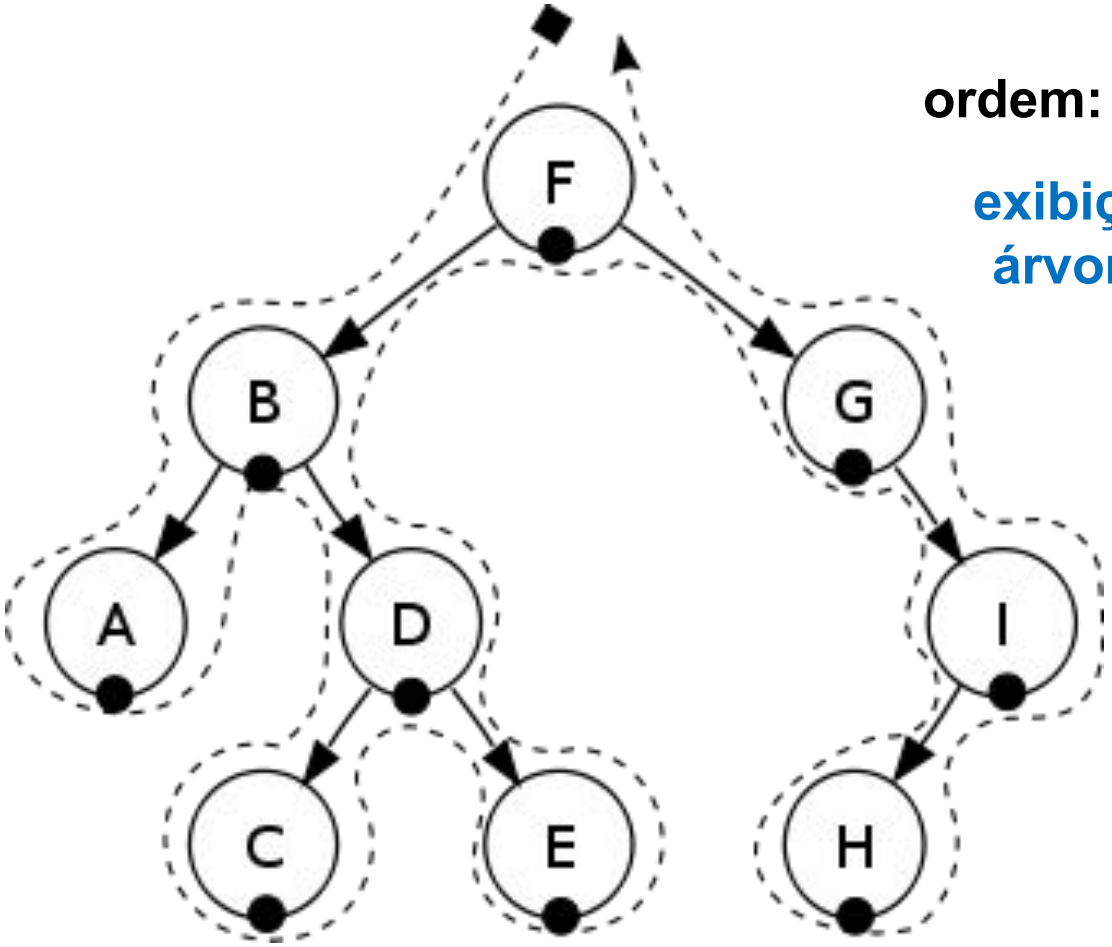
Exemplo: percorrimento simétrico



ordem: A, B, C, D, E, F, G, H, I

fonte: wikipedia

Exemplo: percorrimento simétrico



ordem: A, B, C, D, E, F, G, H, I

exibição de conteúdo em árvore binária de busca

fonte: wikipedia

Árvore binária: percorrimento

Busca em profundidade:

Pré-ordem: trata nó raiz, percorre subárvore a esquerda, e percorre subárvore a direita

Ex: exibir o conteúdo de uma árvore (notação textual)

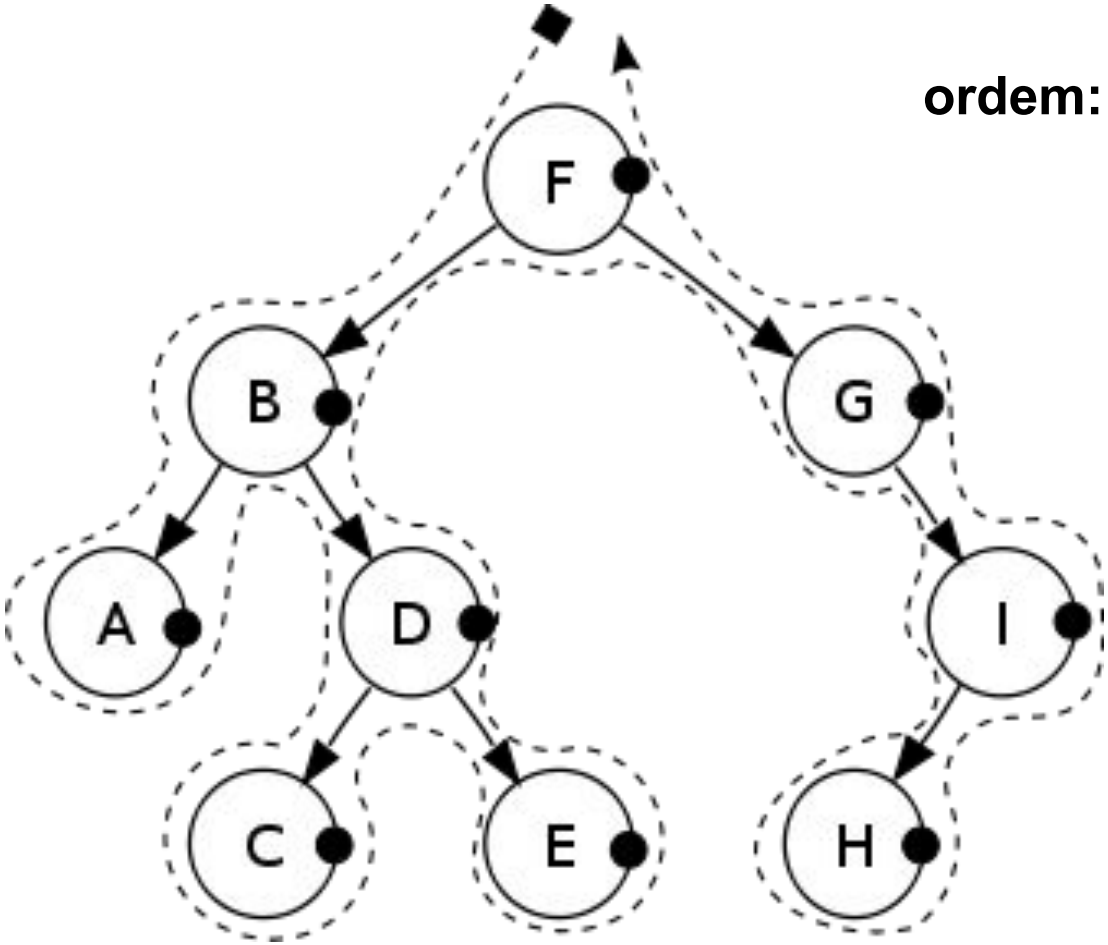
Em-ordem (ordem simétrica): percorre subárvore a esquerda, trata nó raiz, e percorre subárvore a direita

Ex: exibir uma expressão matemática representada em árvore

Pós-ordem: percorre subárvore a esquerda, percorre subárvore a direita, e trata raiz

Ex: liberação de uma árvore

Exemplo: percorrimento pós-ordem



ordem: A, C, E, D, B, H, I, G, F

fonte: wikipedia

Árvore binária: percorrimento

Busca em largura:

Percorrimento dos nós da árvore por nível

Árvore binária: percorrimento

Busca em largura:

Percorrimento dos nós da árvore por nível

Trata o nó raiz, depois seus filhos, seus netos e assim por diante, até não haver mais descendentes

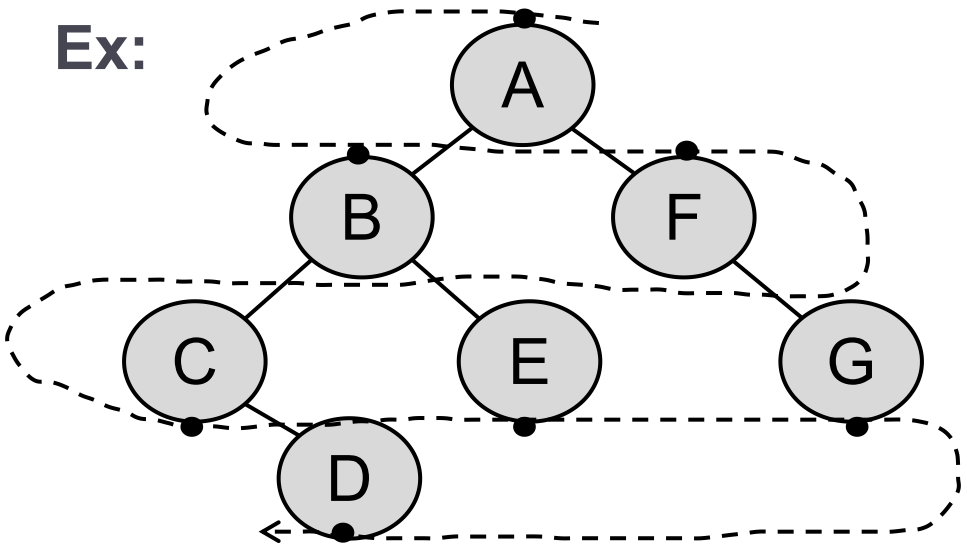
Árvore binária: percorrimento

Busca em largura:

Percorrimento dos nós da árvore por nível

Trata o nó raiz, depois seus filhos, seus netos e assim por diante, até não haver mais descendentes

Ex:



ordem: A, B, F, C, E, G, D

Árvores binárias: operações básicas

- Criar uma árvore
  - Vazia (nula)
  - Não nula
- Liberar uma árvore
- Inserir um novo nó
  - Nó raiz
  - Nó folha
- Buscar um nó
- Remover um nó folha
- Percorrer a árvore

Árvores binárias: operações básicas

- Criar uma árvore
  - Vazia (nula)
  - Não nula
- Liberar uma árvore
- Inserir um novo nó
  - Nó raiz
  - Nó folha
- Buscar um nó
- Remover um nó folha
- Percorrer a árvore

caso particular do
criar árvore

Especificação do TAD AB

Cabeçalho:

Dados: número inteiro

Lista de operações:

cria_vazia: cria uma árvore binária vazia

cria_arvore: cria uma árvore binária não vazia

arvore_vazia: verifica se a árvore está vazia

libera_arvore: libera o espaço de memória alocado para uma árvore, tornando-a vazia

busca: verifica se um dado elemento pertence a árvore

remove_folha: remove o nó cujo valor é dado se ele for folha

exibe_arvore: um exemplo da operação de **percorrimento** de uma árvore. Neste caso, iremos percorrer a árvore e imprimir cada nó.

Especificação do TAD AB

Operação ***cria_vazia:***

Entrada: nenhuma

Pré-condição: nenhuma

Processo: coloca a árvore binária no **estado de vazia**.

Saída: o endereço de uma árvore vazia

Pós-condição: nenhuma

Especificação do TAD AB

Operação ***cria_arvore:***

Entrada: o elemento do nó raiz e os endereços das subárvores à esquerda (SAE) e à direita (SAE)

Pré-condição: nenhuma

Processo: cria um nó raiz cujo valor é do elemento, faz seu filho à esquerda ser a raiz da SAE e seu filho à direita ser a raiz da SAD.

Saída: o endereço da nova árvore binária

Pós-condição: uma nova árvore binária criada

Especificação do TAD AB

Operação ***arvore_vazia:***

Entrada: endereço da árvore

Pré-condição: nenhuma

Processo: verifica se a árvore binária está no **estado de vazia**

Saída: 1 - se vazia ou 0 - caso contrário

Pós-condição: nenhuma

Especificação do TAD AB

Operação ***libera_arvore*:**

Entrada: endereço do endereço da árvore a ser liberada

Pré-condição: a árvore não estar vazia

Processo: percorre a árvore liberando o espaço alocado para cada nó, até que a árvore volte para o estado de vazia.

Saída: nenhuma

Pós-condição: árvore de entrada no estado de vazia

Especificação do TAD AB

Operação ***busca:***

Entrada: endereço da árvore e o elemento a ser encontrado

Pré-condição: a árvore não estar vazia

Processo: percorrer a árvore até encontrar o nó ou não haver mais nós para visitar.

Saída: 1 - se o elemento existe ou 0 - caso contrário

Pós-condição: nenhuma

Especificação do TAD AB

Operação ***remove_folha:***

Entrada: endereço do endereço da árvore (**referência**) e o elemento a ser removido

Pré-condição: a árvore não estar vazia

Processo: percorre a árvore até encontrar o nó ou não haver mais nós para visitar. Se encontrar o nó e ele for folha, remova-o da árvore binária.

Saída: 1- se operação bem sucedida ou 0 - caso contrário

Pós-condição: árvore de entrada com um nó folha a menos

Especificação do TAD AB

Operação ***exibe_arvore:***

Entrada: endereço da árvore a ser exibida

Pré-condição: nenhuma

Processo: caminhe pela árvore em pré-ordem, apresentando o valor de cada nó.

Saída: nenhuma

Pós-condição: nenhuma

Árvores binárias: formas de implementação

Implementações estáticas:

Baseada em vetor (**contiguidade física**)
Baseada em tabela

Árvores binárias: formas de implementação

Implementações estáticas:

Baseada em vetor (**contiguidade física**)
Baseada em tabela

Vantagem:

Acesso direto a qualquer nó

Árvores binárias: formas de implementação

Implementações estáticas:

Baseada em vetor (**contiguidade física**)
Baseada em tabela

Vantagem:

Acesso direto a qualquer nó

Desvantagens:

Necessita determinar o tamanho máximo da árvore
Movimentação de dados na inserção e remoção

Árvores binárias: formas de implementação

Implementações estáticas:

Baseada em vetor (**contiguidade física**)
Baseada em tabela

Vantagem:

Acesso direto a qualquer nó

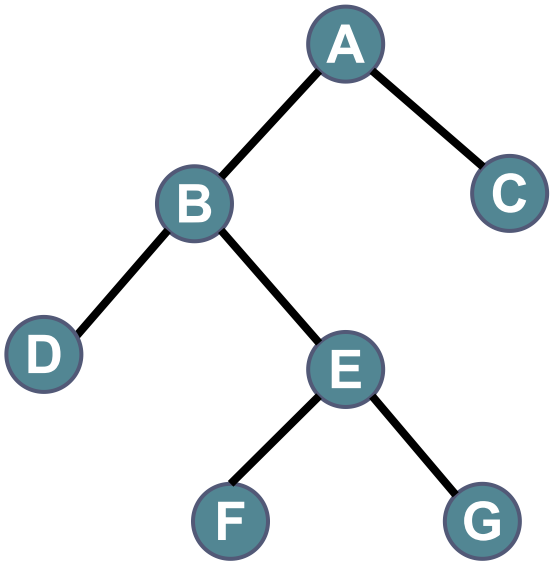
Desvantagens:

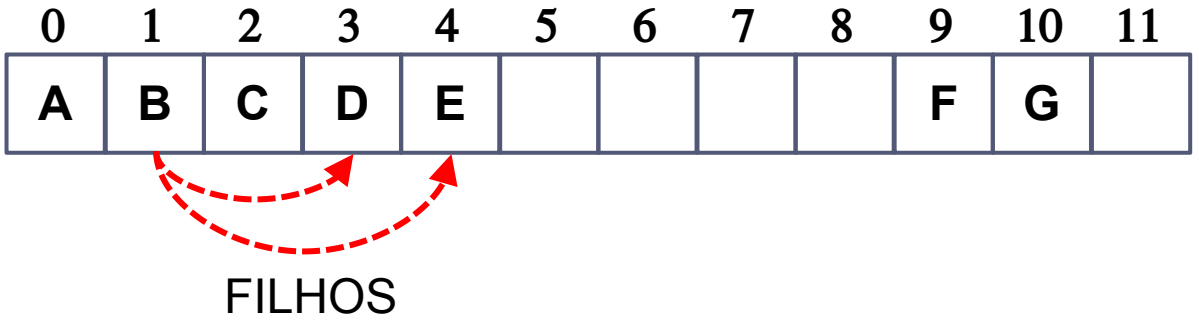
Necessita determinar o tamanho máximo da árvore
Movimentação de dados na inserção e remoção

Implementação **não muito interessante**, pois árvores são **estruturas inerentemente dinâmicas**

Exemplo de implementação

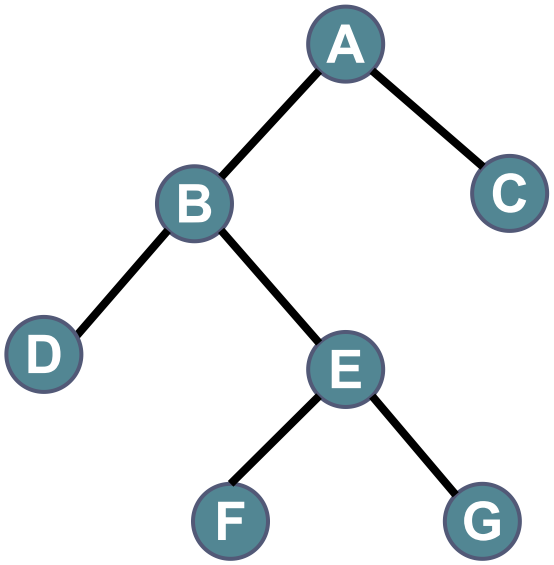
Contiguidade física:



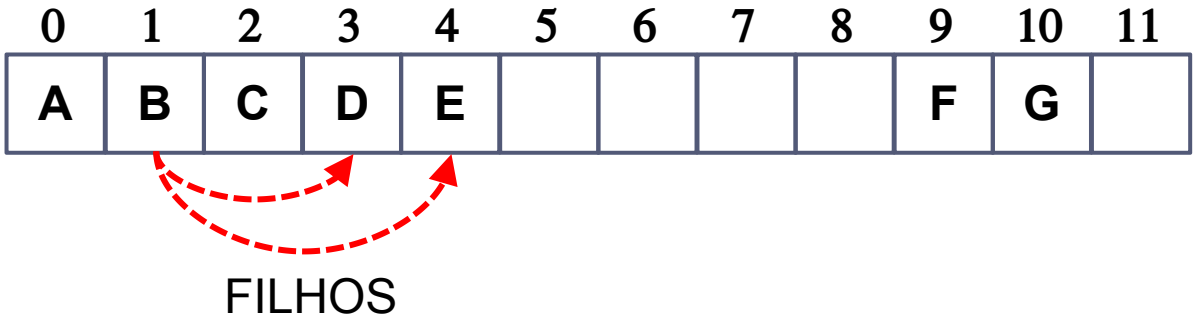


Exemplo de implementação

Contiguidade física:

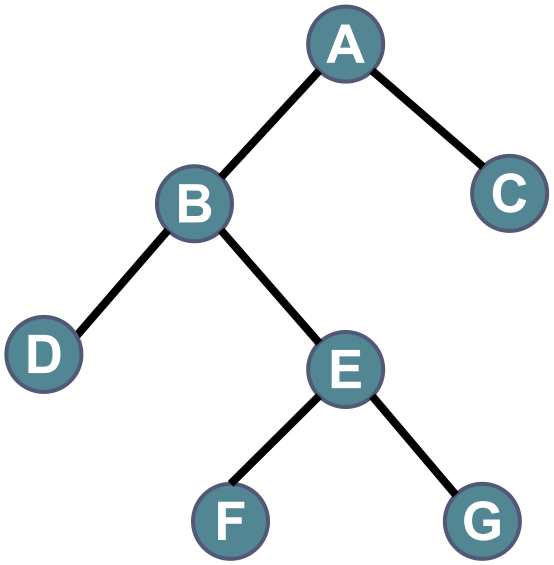


$$FILHO\_ESQ(PAI) = 2 * PAI + 1$$
$$FILHO\_DIR(PAI) = 2 * PAI + 2$$

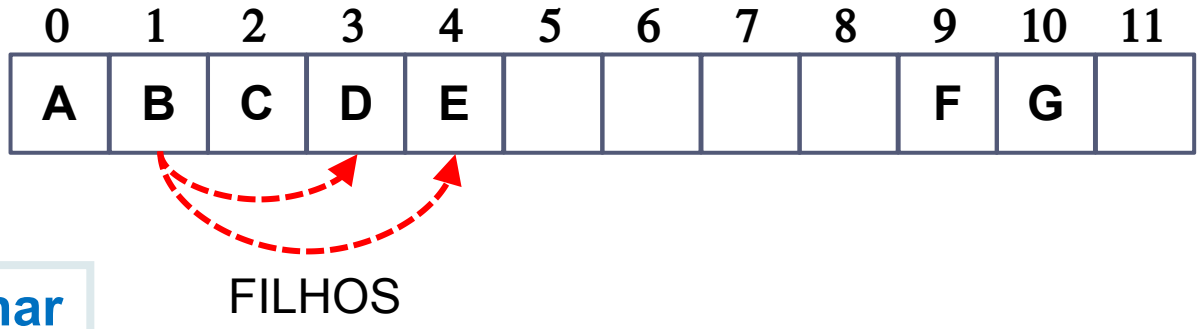


Exemplo de implementação

Contiguidade física:



$$FILHO\_ESQ(PAI) = 2 * PAI + 1$$
$$FILHO\_DIR(PAI) = 2 * PAI + 2$$

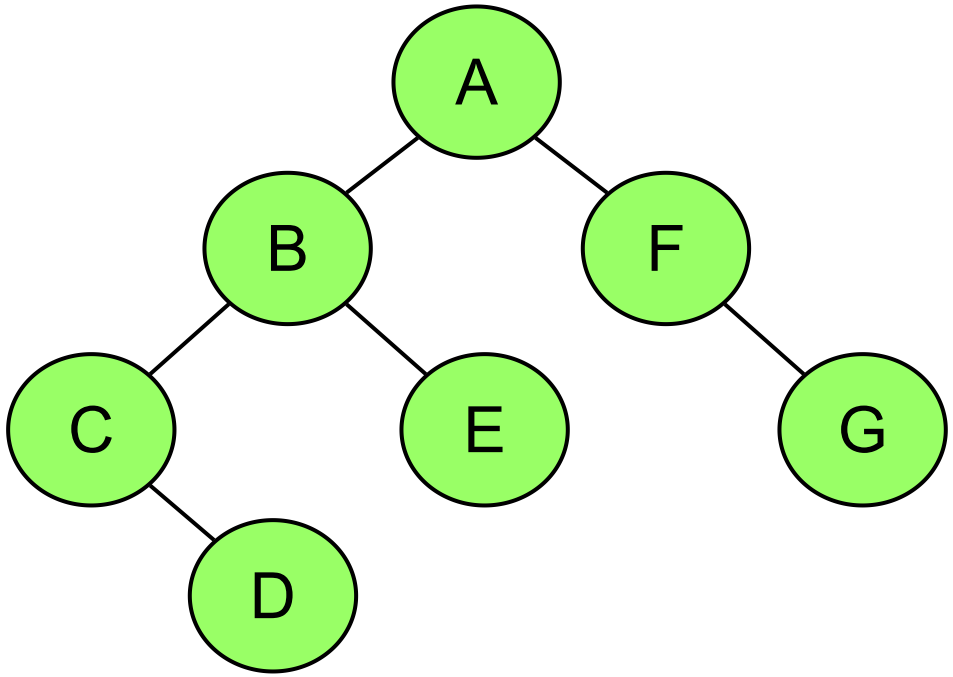


Não necessita armazenar o número de descendentes

Exemplo de implementação

Baseada em tabela:

Índice	Raiz	SAE	SAD
0	A	1	2
1	B	3	4
2	F	-1	5
3	C	-1	6
4	E	-1	-1
5	G	-1	-1
6	D	-1	-1



Árvores binárias: formas de implementação

Implementação dinâmica:

Baseada no **encadeamento dos nós**

Árvores binárias: formas de implementação

Implementação dinâmica:

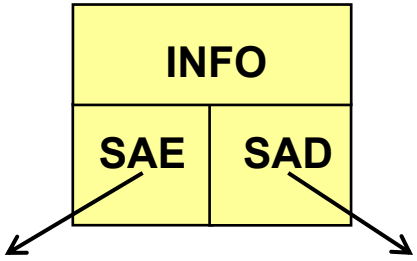
Baseada no **encadeamento dos nós**

Estrutura básica do nó:

Campo **informação** do nó raiz

Endereço da subárvore à esquerda (SAE)

Endereço da subárvore à direita (SAD)



Estrutura Nó

Árvores binárias: formas de implementação

Implementação dinâmica:

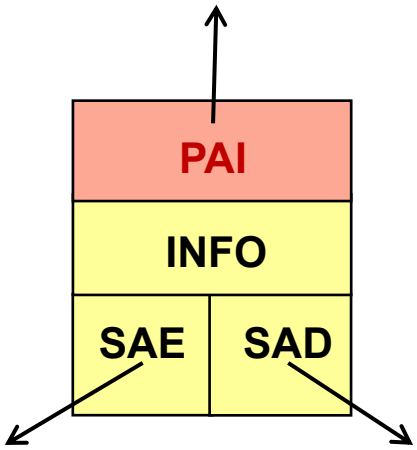
Baseada no **encadeamento dos nós**

Estrutura básica do nó:

Campo **informação** do nó raiz

Endereço da subárvore à esquerda (SAE)

Endereço da subárvore à direita (SAD)



Estrutura Nó

Uma variação menos usual acrescenta à estrutura do nó um **campo que aponta para o nó pai**

Equivalente ao **encadeamento duplo** em listas lineares

Facilita a navegação de baixo para cima

Aumenta o espaço de memória requerido

Árvores binárias: formas de implementação

Implementação dinâmica:

Vantagens:

- Otimização da memória alocada
- Facilidade de manipulação
- Simplificação da implementação de operações

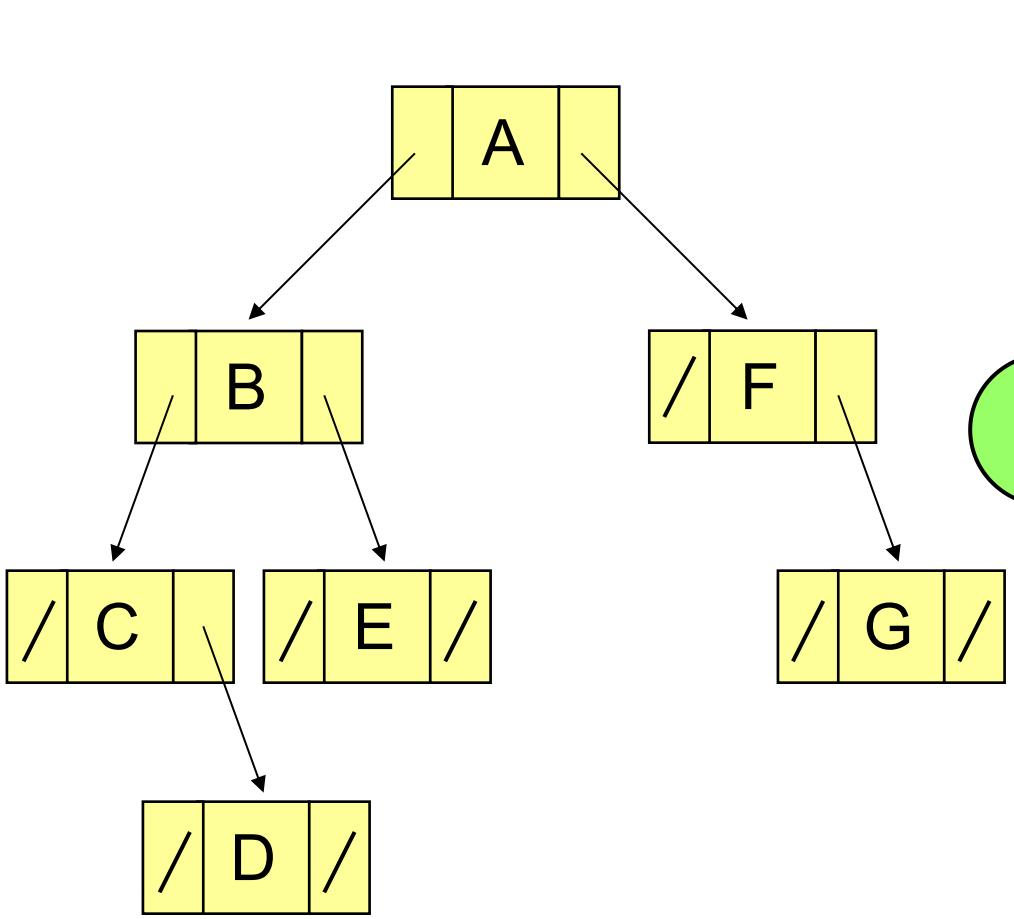
Desvantagem:

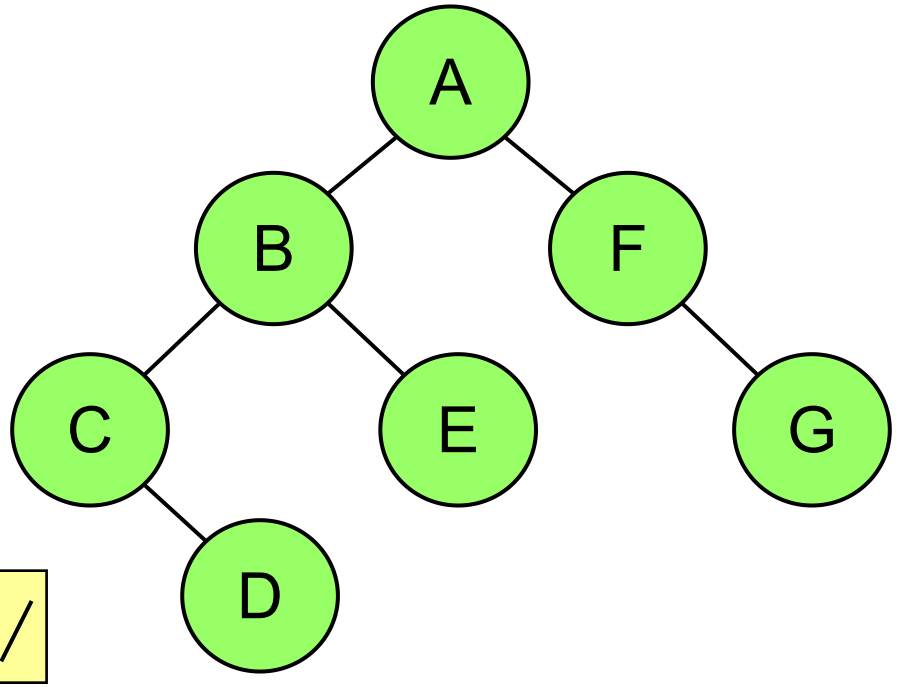
- Acesso direto apenas ao nó raiz

Indicada para **estruturas dinâmicas** com frequentes inserções e remoções

Exemplo de implementação

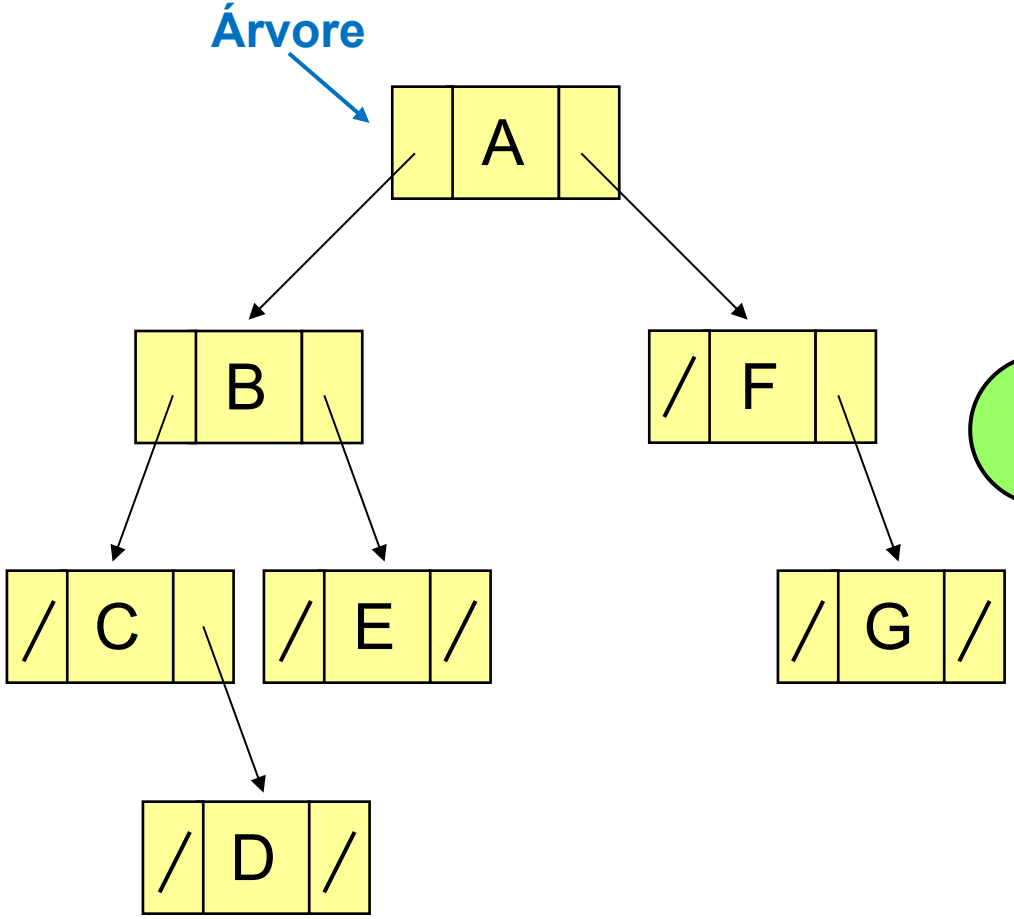
Encadeamento de nós (estrutura básica):

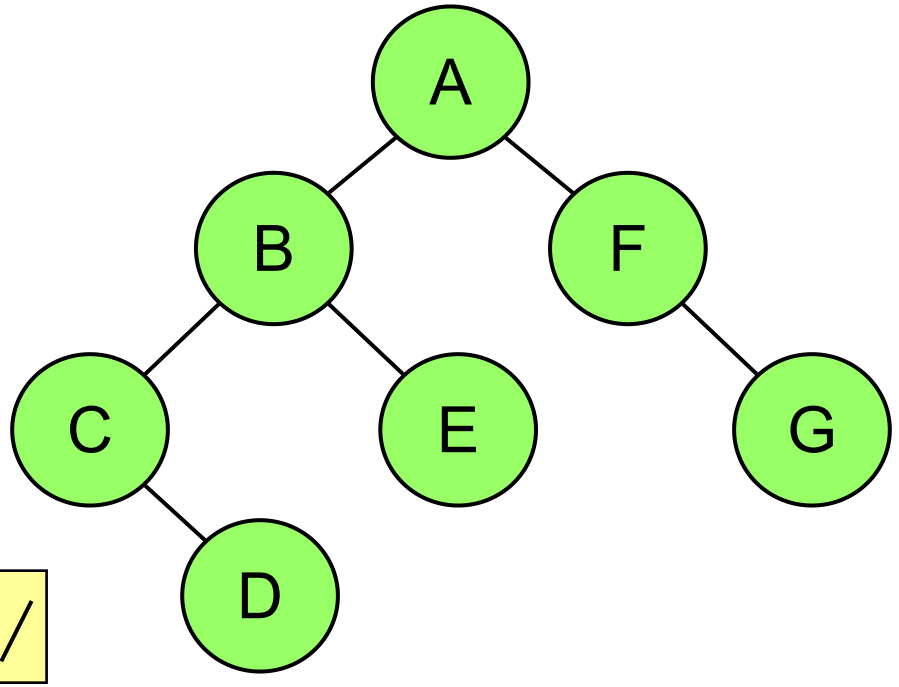




Exemplo de implementação

Encadeamento de nós (estrutura básica):

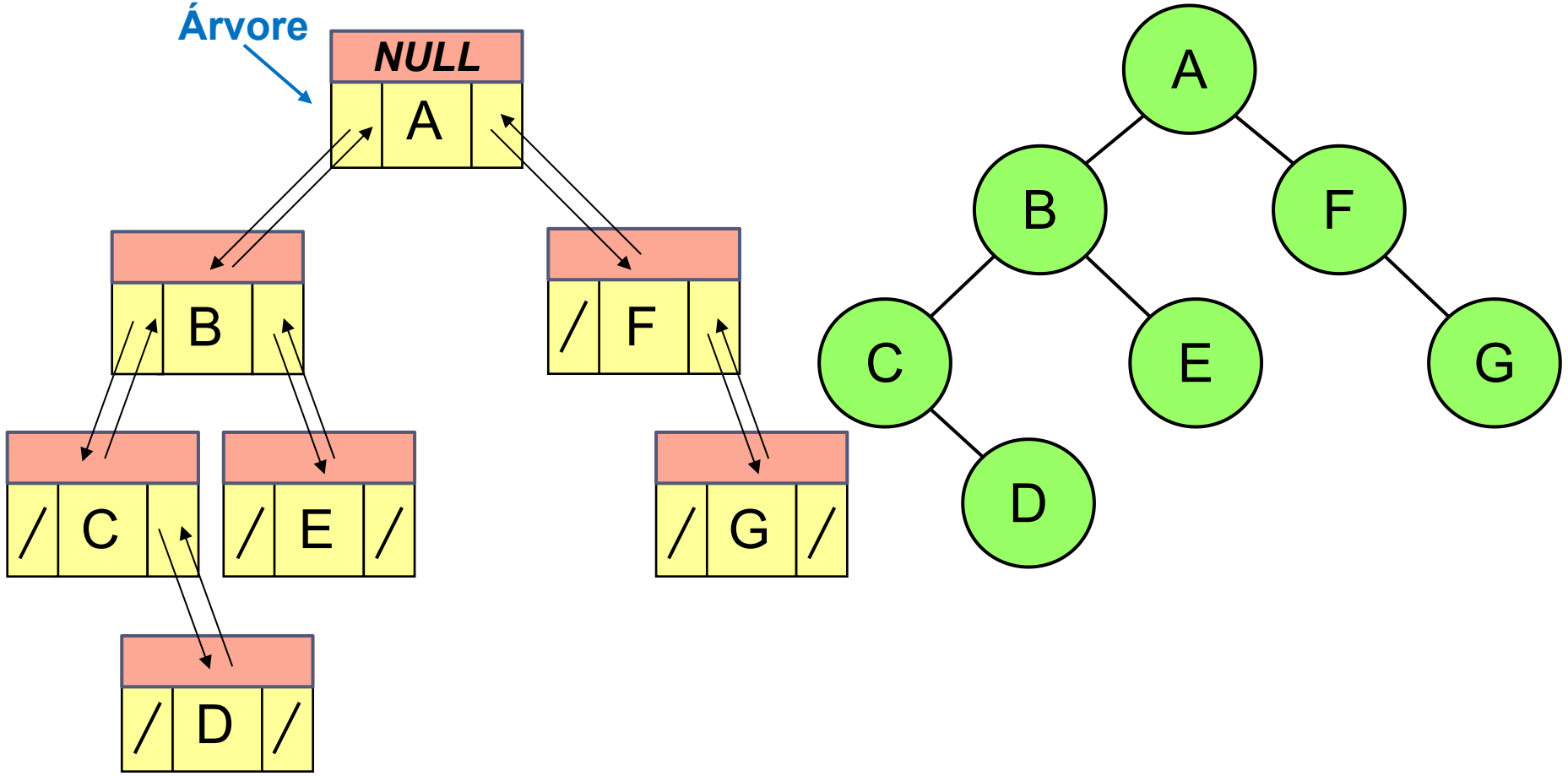




Árvore é um ponteiro para o nó raiz

Exemplo de implementação

Encadeamento de nós (estrutura c/ nó pai):



Árvore binária: estrutura de representação

Estrutura do nó:

Campo **INFO**: informações do nó raiz (**qualquer tipo**)

Campo **SAE**: endereço da subárvore à esquerda

Campo **SAD**: endereço da subárvore à direita

Estrutura Nó

Árvore binária: estrutura de representação

Estrutura do nó:

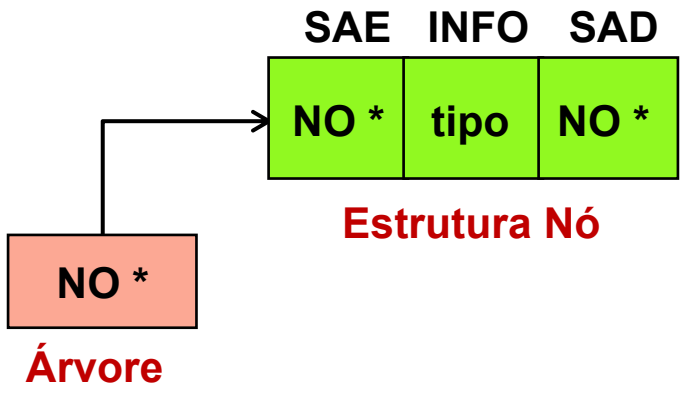
Campo **INFO**: informações do nó raiz (**qualquer tipo**)

Campo **SAE**: endereço da subárvore à esquerda

Campo **SAD**: endereço da subárvore à direita

Árvore: endereço do nó raiz

Ponteiro para o tipo nó



Árvore binária: estrutura de representação

Implementação em C:

```
// Estrutura de um nó
struct no {
    int info;
    struct no *sae,
    struct no *sad;
};
```

Árvore binária: estrutura de representação

Implementação em C:

```
// Estrutura de um nó
struct no {
    int info;
    struct no *sae,
    struct no *sad;
};

// Árvore
typedef struct no * Arv;
```

Árvore binária: estrutura de representação

Implementação em C:

```
// Estrutura de um nó
struct no {
    int info;
    struct no *sae,
    struct no *sad;
};
```

arvBin.c

```
// Árvore
typedef struct no * Arv;
```

arvBin.h

Especificação do TAD AB

Operação ***cria_vazia:***

Entrada: nenhuma

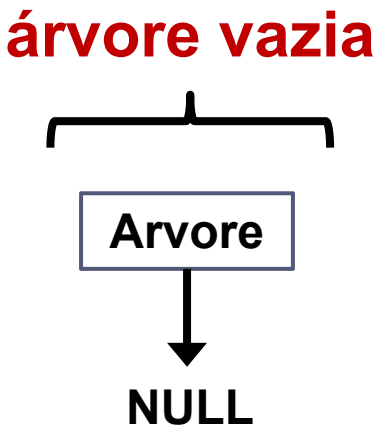
Pré-condição: nenhuma

Processo: coloca a árvore binária no **estado de vazia**.

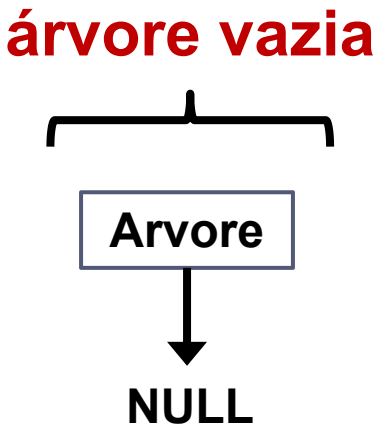
Saída: o endereço de uma árvore vazia

Pós-condição: nenhuma

Árvore binária: implementação



Árvore binária: implementação



Arv ***cria_vazia*** *()*

retorna NULL;

FIM

Especificação do TAD AB

Operação ***cria_arvore:***

Entrada: o elemento do nó raiz e os endereços das subárvores à esquerda (SAE) e à direita (SAE)

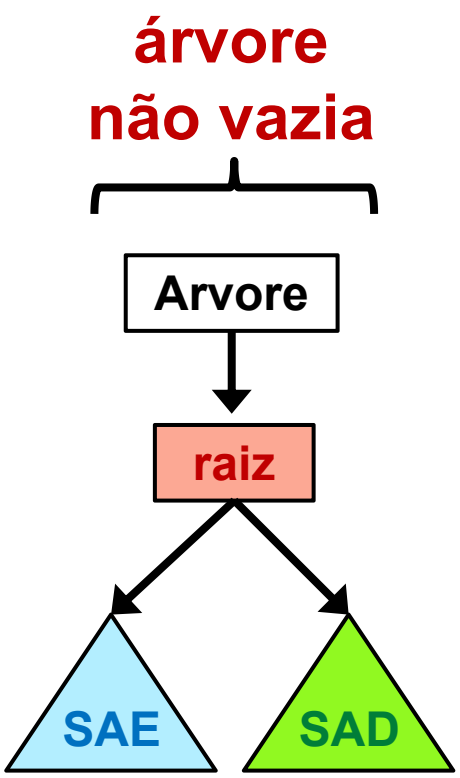
Pré-condição: nenhuma

Processo: cria um nó raiz cujo valor é do elemento, faz seu filho à esquerda ser a raiz da SAE e seu filho à direita ser a raiz da SAD.

Saída: o endereço da nova árvore binária

Pós-condição: uma nova árvore binária criada

Árvore binária: implementação



Arv ***cria_arvore*** *(int elem, Arv esq, Arv dir)*

Aloca um novo nó;

SE *falha na alocação* ***ENTÃO***

retorna ***NULL****;*

FIM_SE

Campo info do novo nó = elem;

Campo sae do novo nó = esq;

Campo sad do novo nó = dir;

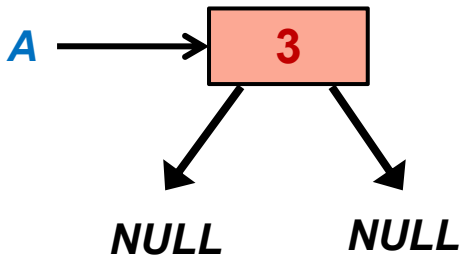
retorna endereço do novo nó;

FIM

Exemplos de utilização da função

Criação de uma AB com um único nó (raiz = 3):

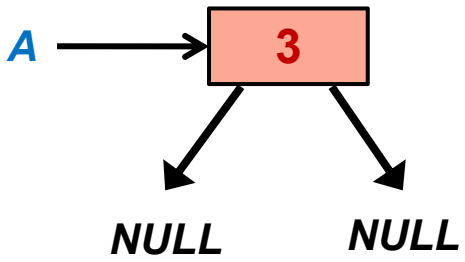
*Arv **A** = **cria_arvore** (3, NULL, NULL);*



Exemplos de utilização da função

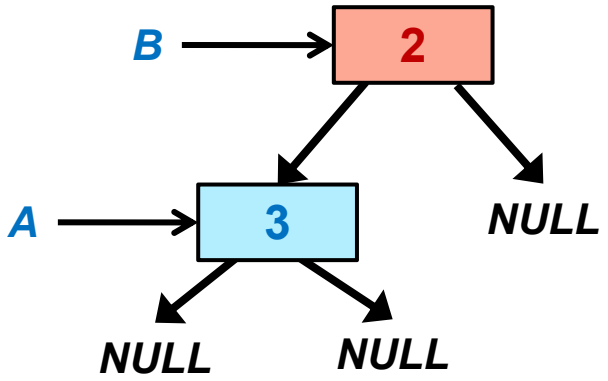
Criação de uma AB com um único nó (raiz = 3):

*Arv **A** = **cria_arvore** (3, NULL, NULL);*



Inserção de um nó como raiz (raiz = 2):

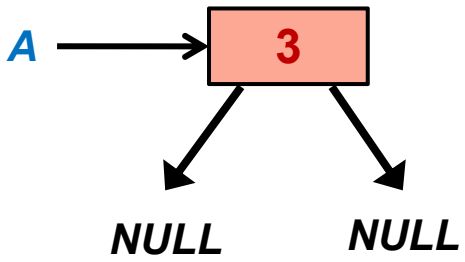
*Colocando **A** como subárvore à esquerda:*
*Arv **B** = **cria_arvore** (2, A, NULL);*



Exemplos de utilização da função

Criação de uma AB com um único nó (raiz = 3):

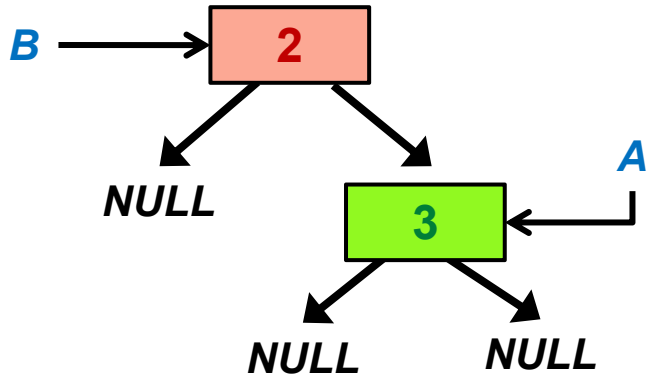
*Arv **A** = **cria_arvore** (3, NULL, NULL);*



Inserção de um nó como raiz (raiz = 2):

*Colocando **A** como subárvore à esquerda:*
*Arv **B** = **cria_arvore** (2, A, NULL);*

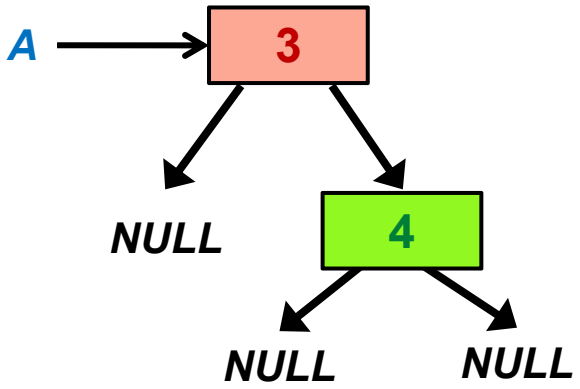
*Colocando **A** como subárvore à direita:*
*Arv **B** = **cria_arvore** (2, NULL, A);*



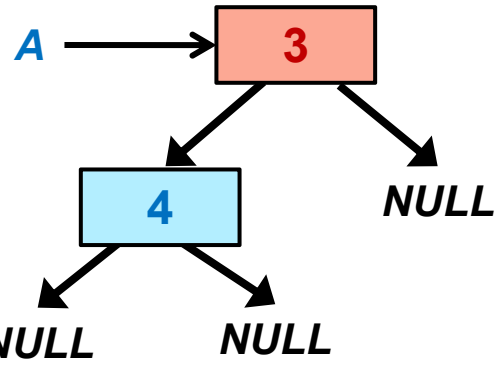
Exemplos de utilização da função

Inserção de um nó como folha (folha = 4):

*Colocando o **novo nó** como **SAD de A**:*
***A->SAD** = **cria_arvore** (4, NULL, NULL);*



*Colocando o **novo nó** como **SAE** de A:*
***A->SAE** = **cria_arvore** (4, NULL, NULL);*



Especificação do TAD AB

Operação ***arvore_vazia:***

Entrada: endereço da árvore

Pré-condição: nenhuma

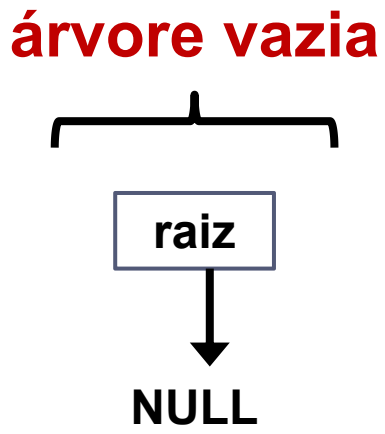
Processo: verifica se a árvore binária está no **estado de vazia**

Saída: 1 - se vazia ou 0 - caso contrário

Pós-condição: nenhuma

Árvore binária: implementação

```
arvore_vazia (Arv A)
    SE A = NULL ENTÃO
        retorna 1;
    SENÃO
        retorna 0;
    FIM_SE
FIM
```



Especificação do TAD AB

Operação ***libera_arvore*:**

Entrada: endereço do endereço da árvore a ser liberada

Pré-condição: a árvore não estar vazia

Processo: percorre a árvore liberando o espaço alocado para cada nó, até que a árvore volte para o estado de vazia.

Saída: nenhuma

Pós-condição: árvore de entrada no estado de vazia

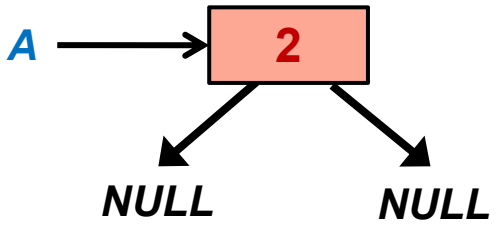
Árvore binária: implementação

```
libera_arvore (Arv * A)
    SE árvore não estiver vazia ENTÃO
        Libera subárvore à esquerda;
        Libera subárvore à direita;
        Libera memória alocada para o nó raiz; // free(*A);
    FIM_SE

    Faz conteúdo de A = NULL; // *A = NULL;
FIM
```

Exemplos de utilização da função

Liberação de uma AB com um único nó:

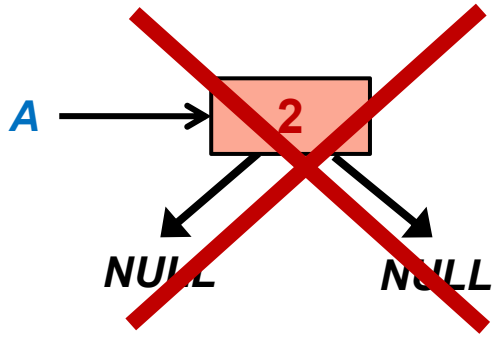
libera_arvore (&A);



Exemplos de utilização da função

Liberação de uma AB com um único nó:

libera_arvore (&A);

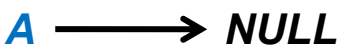


free(A);

Exemplos de utilização da função

Liberação de uma AB com um único nó:

libera_arvore (&A);

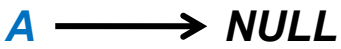


A = NULL;

Exemplos de utilização da função

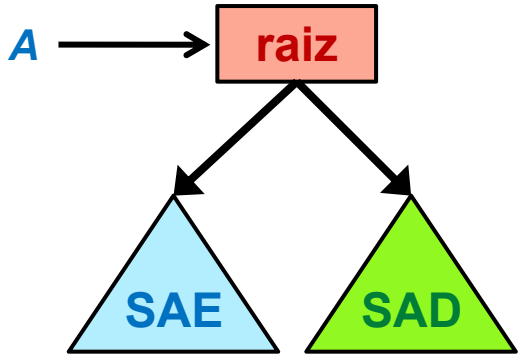
Liberação de uma AB com um único nó:

libera_arvore (&*A*);



Liberação de uma AB com mais de um nó:

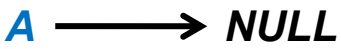
libera_arvore (&*A*);



Exemplos de utilização da função

Liberação de uma AB com um único nó:

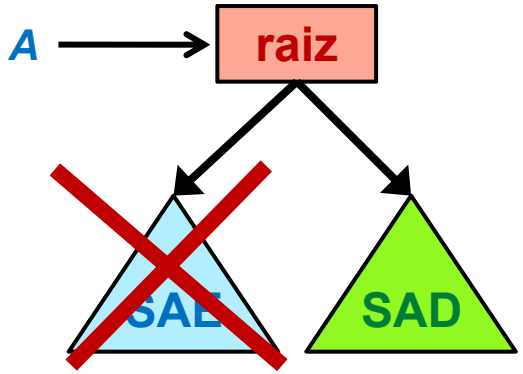
libera_arvore (&A);



Liberação de uma AB com mais de um nó:

libera_arvore (&A);

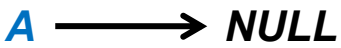
libera_arvore(&(A->sae));



Exemplos de utilização da função

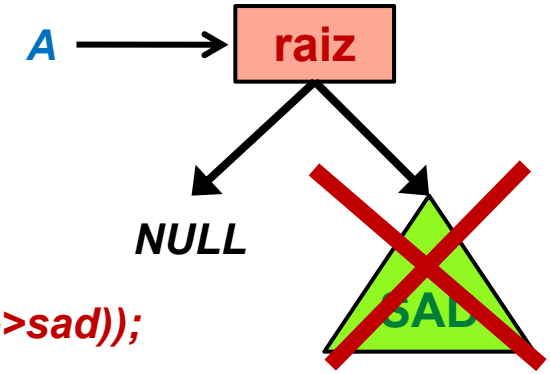
Liberação de uma AB com um único nó:

libera_arvore (&A);



Liberação de uma AB com mais de um nó:

libera_arvore (&A);

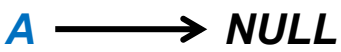


libera_arvore(&(A->sad));

Exemplos de utilização da função

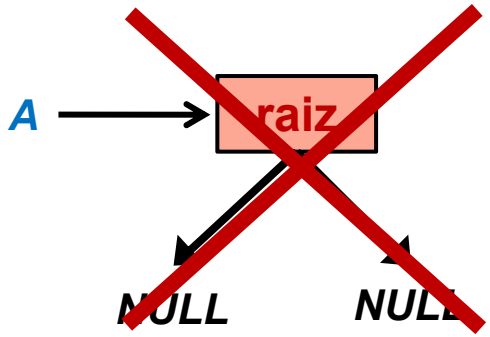
Liberação de uma AB com um único nó:

libera_arvore (&A);



Liberação de uma AB com mais de um nó:

libera_arvore (&A);

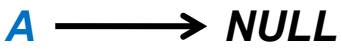


free(A);

Exemplos de utilização da função

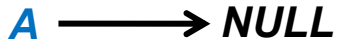
Liberação de uma AB com um único nó:

libera_arvore (&A);



Liberação de uma AB com mais de um nó:

libera_arvore (&A);



A = NULL;

Especificação do TAD AB

Operação ***busca:***

Entrada: endereço da árvore e o elemento a ser encontrado

Pré-condição: a árvore não estar vazia

Processo: percorrer a árvore até encontrar o nó ou não haver mais nós para visitar.

Saída: 1 - se o elemento existe ou 0 - caso contrário

Pós-condição: nenhuma

Árvore binária: implementação

int ***busca*** *(Arv A, int elem)*

SE *árvore vazia* ***ENTÃO***
retorna 0;
FIM_SE

SE *campo* ***info*** *de A = elem* ***ENTÃO***
retorna 1;
FIM_SE

Busca elem na SAE de A;
SE *encontrou* ***ENTÃO***
retorna 1;
FIM_SE

...

...

Busca elem na SAD de A;
SE *encontrou* ***ENTÃO***
retorna 1;
FIM_SE

retorna 0;

FIM

Especificação do TAD AB

Operação ***remove_folha:***

Entrada: endereço do endereço da árvore (**referência**) e o elemento a ser removido

Pré-condição: a árvore não estar vazia

Processo: percorre a árvore até encontrar o nó ou não haver mais nós para visitar. Se encontrar o nó e ele for folha, remova-o da árvore binária.

Saída: 1- se operação bem sucedida ou 0 - caso contrário

Pós-condição: árvore de entrada com um nó folha a menos

Árvore binária: implementação

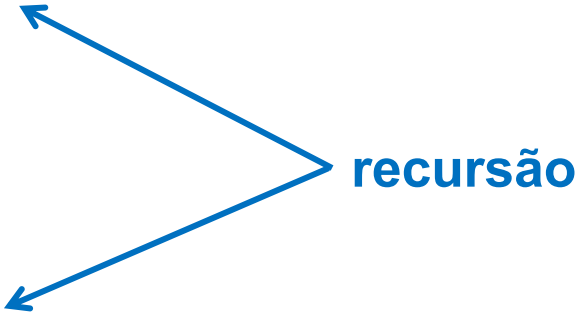
```
int remove_folha (Arv * A, int elem)
    SE árvore vazia ENTÃO
        retorna 0;
    FIM_SE
    // Trata nó raiz
    SE campo info do nó raiz = elem ENTÃO
        SE SAE da raiz = NULL E SAD da raiz = NULL ENTÃO
            Libera memória alocada para o nó raiz;
            Coloca a árvore no estado de vazia (árvore = NULL);
            retorna 1;
        SENÃO
            retorna 0;  // Não é folha
        FIM_SE
    FIM_SE
...
```

Árvore binária: implementação

```
...
   SENÃO  // Nó raiz ≠ elem
      Remove folha da subárvore a esquerda = elem;
      SE operação bem sucedida ENTÃO
         retorna 1;
      FIM_SE

      Remove folha da subárvore a direita = elem;
      SE operação bem sucedida ENTÃO
         retorna 1;
      FIM_SE

      retorna 0;
   FIM_SE
FIM
```

Árvore binária: implementação

```
...
   SENÃO  // Nó raiz ≠ elem
      Remove folha da subárvore a esquerda = elem;
      SE operação bem sucedida ENTÃO
         retorna 1;
      FIM_SE

      Remove folha da subárvore a direita = elem;
      SE operação bem sucedida ENTÃO
         retorna 1;
      FIM_SE

      retorna 0;
   FIM_SE
FIM
```



Especificação do TAD AB

Operação ***exibe_arvore:***

Entrada: endereço da árvore a ser exibida

Pré-condição: nenhuma

Processo: caminhe pela árvore em pré-ordem, apresentando o valor de cada nó.

Saída: nenhuma

Pós-condição: nenhuma

Árvore binária: implementação

```
exibe_arvore (Arv A)

    SE árvore vazia ENTÃO
        escreva("<>");
    FIM_SE

    escreva ("< ");   // Abertura de contexto na notação textual

    Exibe o campo info de A;
    Exibe subárvore a esquerda de A;
    Exibe subárvore a direita de A;

    escreva(">");   // Fechamento de contexto na notação textual

FIM
```

Árvore binária: implementação

exibe_arvore *(Arv A)*

SE *árvore vazia* ***ENTÃO***
escreva("<>");
FIM_SE

escreva ("< "); // Abertura de contexto na notação textual

Exibe o campo ***info*** *de A;*
Exibe subárvore a esquerda de A;
Exibe subárvore a direita de A;
} **recursão**

escreva(">"); // Fechamento de contexto na notação textual

FIM

Árvore binária: implementação

- Árvores são estruturas inerentemente recursivas
  - Recursão **facilita a implementação** das operações
  - Funções utilizam a **pilha de recursão**

Árvore binária: implementação

Árvores são estruturas inerentemente recursivas

- Recursão **facilita a implementação** das operações
- Funções utilizam a **pilha de recursão**

Implementação iterativa:

- Uso de estruturas **pilha ou fila explícitas**
- Substituição das chamadas recursivas por **estruturas de repetição (laços)**

Árvore binária: implementação

Árvores são estruturas inerentemente recursivas

- Recursão **facilita a implementação** das operações
- Funções utilizam a **pilha de recursão**

Implementação iterativa:

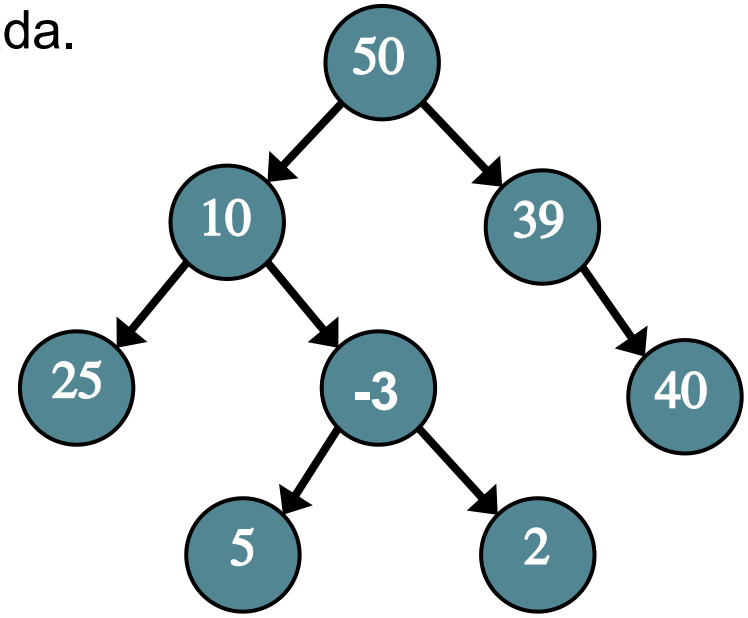
- Uso de estruturas **pilha ou fila explícitas**
- Substituição das chamadas recursivas por **estruturas de repetição (laços)**

Recursividade *vs.* Iteratividade

Simplificação *vs.* Desempenho

Exercícios

1. Apresente a árvore através das representações de contiguidade física, tabela e encadeada.



2. Implemente as operações especificadas para o TAD árvore binária de números inteiros. Complemente o conjunto de operações com 2 operações: uma que determina a altura de uma árvore e outra que retorna o endereço do nó pai de um dado elemento (se existente)

Bibliografia

Slides adaptados do material da Profa. Dra. Gina Maira Barbosa de Oliveira, da Profa. Dra. Denise Guliato e do Prof. Dr. Bruno Travençolo.

EDELWEISS, N; GALANTE, R. Estruturas de dados (Série Livros Didáticos Informática UFRGS, v. 18), Bookman, 2008.

CORMEN, T.H. et al. Algoritmos: Teoria e Prática, Campus, 2002

ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (2ª ed.), Thomson, 2004

CELES, W.; CERQUEIRA, R. & RANGEL, J. L. Introdução a Estruturas de Dados: com técnicas de programação em C, Elsevier, 2004